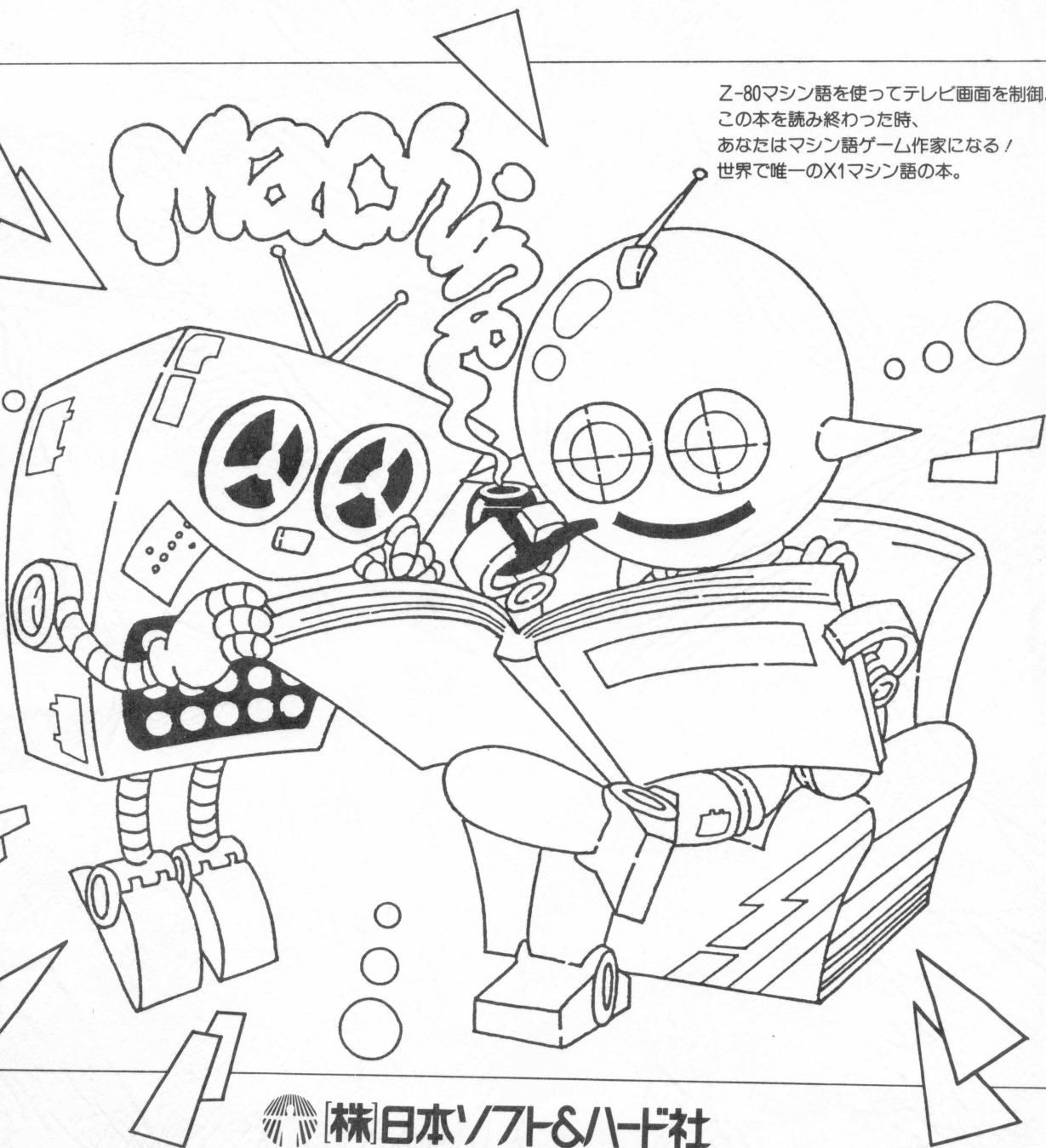
清水 保弘 著

マシン語入門

Z-80マシン語を使ってテレビ画面を制御。
この本を読み終わった時、
あなたはマシン語ゲーム作家になる！
世界で唯一のX1マシン語の本。

(株)日本ソフト&ハード社

まえがき

シャープＸ１の全てのユーザーにこの本を贈ります。

Ｘ１ユーザーの皆さんの家の本棚には、多くのマイコン雑誌と並んで、何冊かの「パソコン入門」的な本が置かれていると思います。皆さんは、これらの中で、最後まできちんと読み通した本は何冊ありますか？　おそらく一冊もないのではないでしょうか。今まで市販されていたパソコン関係の本の多くは、どれも記述が難かしく、初心者にとって理解に苦しむ点が多いものばかりでした。それでも向上心おうせいな皆さんは、マニュアル等を参考にして、どうにかこうにかＢＡＳＩＣのコマンドの使い方くらいは身に付けられたでしょう。

ＢＡＳＩＣは実に親切な言語ですから、詳しい入門書などなくても、こちらが間違いをおかせばＢＡＳＩＣがちゃんとそれを指摘してくれます。ＢＡＳＩＣが皆さんの先生がわりをしてくれたわけですね。やがて卒業の日が来ました。そしてマシン語入門………。ところが、ここで皆さんは大きな壁に突き当たることになるのです。

今まではＢＡＳＩＣが先生がわりを務めてくれていましたが、今度は、皆さん自身が先生になってコンピューターに仕事の手順を教えてやる立場に立たされたわけなのです。

さあ大変！　皆さんはコンピューターに仕事の仕方を正しく教えられますか？　マシン語の場合、エラー表示などというものはありませんから、皆さんが一か所でも間違えた教え方をすると、Ｘ１のＺ８０ＣＰＵはすぐにヘソを曲げてしまいます。ヘソを曲げたＺ８０は、時には皆さんの先生であるＢＡＳＩＣに八つ当たりをして、ＢＡＳＩＣシステムをメチャメチャに壊してしまうことさえあります。これでは、さすがの皆さんでも詳しい教科書が必要になりますね。

そこで、ヘソ曲がりのＺ８０のなだめ方を、皆さんにだけ、こっそりお教えするのがこの本の役目です。Ｘ１のＺ８０ＣＰＵは、この本を読んだアナタの言うことなら、ヘソなど曲げずに素直に聞いてくれることでしょう。そして今度は、アナタと一緒にゲームをして楽しもうと提案してくるでしょう。Ｚ８０ＣＰＵは、本当は素直でいい子ばかりなのです。どうか、このカワイイＺ８０を末長く可愛がってあげて下さい。

この本は、マシン語初心者のマシン語初心者によるマシン語初心者のためのマシン語入門書です。初心者にとってわかりにくいと思われる所は、最大限にページをさいて詳しい解説を加えてあります。この本を読んでいただければ、今まで何となく良くわからずにウヤムヤになっていた部分がきっと明らかになり、アナタのパソコンライフの未来もバラ色に輝くことでしょう。Ｚ８０ＣＰＵ同様、どうかこの本も、皆さんのパソコンライフの一助としてご活用下さい。

昭和５９年３月　　　　　　　　　　　　㈱日本ソフト＆ハード社　編集部

目　次

第4章　マシン語ゲームに挑戦！………135

第0章 マシン語初体験！

■第0章 マシン語初体験！

0－1／マシン語を見る

さあ、シャープＸ１で、マシン語の勉強を始めましょう。まず、マシン語とは、どんな姿をしているか見てみましょう。

とはいうものの、どうすればマシン語に出会うことができるのでしょうか？　私たちの手許には、Ｘ１の本体とディスプレイテレビ、それから「マニュアル」しかないと仮定いたします。取りあえず、「マニュアル」を繰って探すことにしましょう。

いかがですか？「マニュアル」には、ＢＡＳＩＣのコマンドの説明ばかり出ていて、なかなか「マシン語」に出会えませんね。

実は、この「マニュアル」内には、何か所か意味のあるマシン語プログラムが出ています。まず１か所目は？　マニュアルの１７５ページ「エディット書式」中の「《例》」と書かれている部分を御覧下さい。

```
:FE00=3E ;A CD 13 00 C9
```

これがマシン語です（等号＝の右側の文字列に注目！）。これは立派な「マシン語プログラム」で「文字Ａを表示する」ものです。

もう１か所は？　マニュアルの１９５ページを御覧下さい。「ＢＡＳＩＣテープのコピー作成方法」の中に、コピー作成プログラムのリストが出ていますね。「なーんだ、ＢＡＳＩＣじゃないか？」と思ってはいけませんよ。５行にわたるＤＡＴＡ文がありますね。ここに何やら不可思議な文字列が、５０個も書かれています。

《ＢＡＳＩＣテープ・コピープログラム内の文字列》

２１、６０、ＦＥ、０１、２０、００、ＣＤ、４１、００、２Ａ
７４、ＦＥ、ＥＤ、４Ｂ、７２、ＦＥ、ＣＤ、４４、００、３Ｅ
０１、ＣＤ、１Ｂ、００、ＦＥ、２０、２０、Ｆ７、２１、６０
ＦＥ、０１、２０、００、ＣＤ、３Ｂ、００、２Ａ、７４、ＦＥ
ＥＤ、４Ｂ、７２、ＦＥ、ＣＤ、３Ｅ、００、Ｃ３、１３、ＦＥ

これは、何をかくそう本格的なマシン語プログラムです！　普通、ＢＡＳＩＣのプログラムをテープに録音するには、ＳＡＶＥ　でよかったのですね。しかし、ＢＡＳＩＣのシステムテープは、この方法ではコピーできません。このマシン語プログラムは、ＢＡＳＩＣではできないこと ―― ＢＡＳＩＣのシステムテープ自体のコピー ―― を可能にするものです。マシン語って偉大ですね！

私たちは、マニュアルから２か所、マシン語を探しました。共通しているのは、何やら変な英数字（数字の０～９とアルファベットのＡ～Ｆ）が２つずつ並んでいることですね。また、１か所目の方には、さらに「ＦＥ００」という４桁の英数字列もありました。これらの正体はいったい何でしょうか。

今はまだ、前者のプログラムで「文字Ａが表示される」理由、後者のプログラムで「システムテープがコピーできる」理由は、きっとチンプンカンプンなことでしょう。しかし本書を読み終えた時、皆様は、これらの不可思議な文字列 ―― マシン語 ―― の正体を知ることでしょう。そして、今まで、無味乾燥にしか見えなかった文字列が、急に生き生きと「意味の光」を放ちはじめるのを経験することでしょう。では、「素晴らしいマシン語の世界」へ向かってスタート！

［注］　Ｘ１，Ｘ１Ｃ，Ｘ１Ｄのマニュアル頁対応表を付録６（２６７頁）に記載しておりますので、ご参照下さい。

０－２／マシン語モニター

前節において、私たちは「マニュアル」を探して、マシン語プログラムの姿を観察いたしました。では、私たちの周囲にあるマシン語は、たったこれだけなのでしょうか？

答は、　ＮＯ　です。実は、私たちはウンザリする位、マシン語を見ることができるのです。

ＢＡＳＩＣを走らせている時、画面に　Ｏｋ　という文字が出て、カーソルが点滅していますね。この状態では、私たちは、ＢＡＳＩＣ言語しか扱うことができません。では、マシン語を扱う（入力したり、実行したり、リストを出したり）にはどのようにすればよいのでしょ

う。

実は、BASICの世界からマシン語の世界に通ずる入口があります。BASICコマンドに　MON　というのがあるのを御存じですか？　マニュアル１０７ページを御覧下さい。

制御をＢＡＳＩＣ内蔵の機械語モニターに移します。

と書いてあります。マシン語に初めて接する方は、MONコマンドは、使ったことがないかもしれません。しかし、これからマシン語の勉強をしていく上で、このコマンドと無縁に過ごすことはできません。本書を通じて、以後何回も登場してきますから、是非使い方を覚えていただきたいと思います。

ものは試し、ともかく実行してみましょう。　MON　とキー•インし、CRキー（リターンキー）を押して下さい。

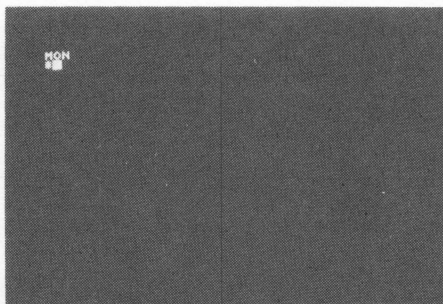



図０－１

普通のBASICコマンドですと、Okが出てからカーソルが点滅するのに、何と星印（＊）の後にカーソルが点滅していますね。どうしたことでしょう！？

これは異常ではありません。私達が、BASICの世界をあとにして、マシン語の世界に入った証拠なのです。

マシン語の世界を監視•制御するプログラムをマシン語（機械語）モニターとよびます。モニター（ｍｏｎｉｔｏｒ）という言葉はよく聞きますが、辞書で引くと、「監視する、制御する」の意味を持つことがわかります。

普通のＢＡＳＩＣ入力状態では、このマシン語モニターは起動されず、「眠って」おります。ＭＯＮというＢＡＳＩＣコマンドは、マシン語モニターを起動するためのものなのです。（以後、マシン語モニターという用語は頻出いたしますので、単にモニターと呼ぶことにします。）

ＭＯＮを実行し、モニターが起動すると、私たちは、ＢＡＳＩＣの手を離れて、モニターの管理下に置かれます。これはＢＡＳＩＣとは全く違う世界ですから、ユーザーにそのことを知らせるための印が星印＊という訳です。この状態のことを、モニターのコマンドレベルともよびます。これに対し、Ｏｋが出てカーソルが点滅している状態をＢＡＳＩＣのコマンドレベルとよびます。

ＢＡＳＩＣのコマンドは、普通の英単語に似せて作られていますね。これに対して、モニターのコマンドは、アルファベット１文字で表わされます。Ｘ１では、次のコマンドが用いられています。

図０－２　《モニターのコマンド》

コマンド名	機　　能
M	メモリーセット
G	ゴーサブ
S	セーブ
L	ロード
V	ベリファイ
D	ダンプ
F	ファインド
T	トランスファ
P	プリンタースイッチ
R	リターン

これら全部をいきなり覚えるのは大変ですから、詳細は、「マニュアル」の１７５～１７７ページに任せることにして、基本的なものから使い方をマスターすることにいたします。

０－３／マシン語を入力する

まず最初にマスターするのは、マシン語の入力方法です。ＭＯＮによりモニターを起動して下さい。＊印が出ましたか？

マシン語の入力には、Ｍコマンドを用います。次の様にキー・インして下さい。

```
*M D000
:D000=3E
:D001=E3
:D002=01
:D003=F4
:D004=31
:D005=ED
:D006=79
:D007=3E
:D008=03
:D009=01
:D00A=F4
:D00B=21
:D00C=ED
:D00D=79
:D00E=C9
:D00F=00
*
```

１行入力するごとにＣＲキーを押すと次の行に進みます。最後に、

：Ｄ００Ｆ＝□

　　　　↑カーソル点滅

となりましたら、 SHIFT ＋ BREAK を押すと再び＊が表示され、モニターのコマンドレベルに戻ります。これがマシン語の入力法です。

まだまだ不明な点が多いことと思いますが、もう少しお待ち下さい。

ここで登場したのは、４桁の英数字（Ｄ０００など）と２桁の英数字（３Ｅなど）ですね。実は、４桁の方はメモリーの番地（アドレス）を表わし、２桁の方がマシン語を表わしています。

これらについてきちんとした説明を与えるには、まずコンピューターの動作原理を理解しな

ければなりません。ちょうどよい機会ですから、次節から３節にわたり、基礎事項を確認することにいたします。

０－４／CPU、ビット、バイト

コンピューターが１つのまとまった動作をするには、大型機・小型機を問わず次の３つの部分が必要です。

中央処理装置（ＣＰＵ）

記憶装置（メモリー）

入出力装置（Ｉ／Ｏ）

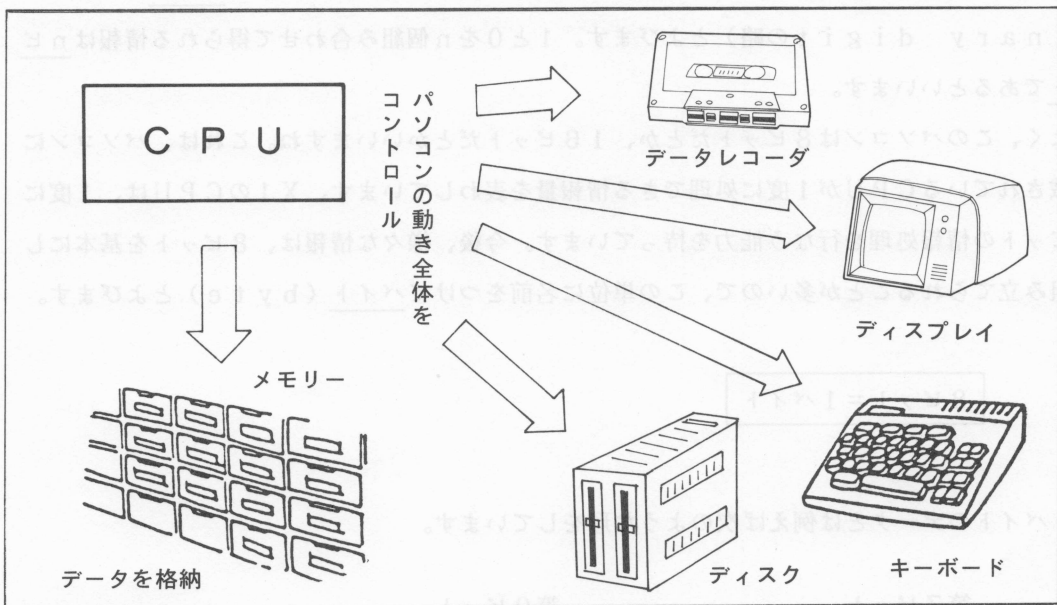



図０－３

まず中央処理装置から見てゆきましょう。英語では、Ｃｅｎｔｒａｌ　Ｐｒｏｃｅｓｓｉｎｇ　Ｕｎｉｔといい、頭文字をとって、普通　ＣＰＵ　とよんでいます。

ＣＰＵは、コンピューターの中核となる部分で「電子頭脳」とも称せられるように、情報の処理・加工、周辺機器の制御などシステム全般をコントロールします。

ＳＦ映画に登場するコンピューターは大型コンピューターで、ＣＰＵの部分は、大きな箱状をしています。しかし、最近の飛躍的な半導体技術の発達は、手のひらに載るほどの大規模集

積回路（ＬＳＩ＝Ｌａｒｇｅ　Ｓｃａｌｅ　Ｉｎｔｅｇｒａｔｉｏｎ）としてＣＰＵを実現することに成功しました。これがマイクロコンピューター（略してマイコン）あるいはマイクロプロセッサーとよばれるものです。

パーソナルコンピューターＸ１の心臓部にもマイコンＬＳＩが搭載されています。

さて、ＣＰＵが処理できる情報というのは電気信号の形をとっています。ある基準電圧より、電圧が高いか低いかというパターンの組み合わせがＣＰＵのわかる言葉です。これを人間にとって理解しやすくするために、普通、高い状態を１、低い状態を０と表記します。この表記法によると、ＣＰＵの処理する情報は、１と０の組み合わせということになります。コンピューターでは、２進法が使われるとよく言いますが、このことを指しているのです。

ＣＰＵが処理できる情報の最小単位（ある信号が１か０かということ）をビット（ｂｉｔ＝ｂｉｎａｒｙ　ｄｉｇｉｔの略）とよびます。１と０をｎ個組み合わせて得られる情報はｎビットであるといいます。

よく、このパソコンは８ビットだとか、１６ビットだとかいいますね。これは、パソコンに搭載されているＣＰＵが１度に処理できる情報量を表わしています、Ｘ１のＣＰＵは、１度に８ビットの情報処理を行なう能力を持っています。今後、様々な情報は、８ビットを基本にして組み立てられることが多いので、この単位に名前をつけてバイト（ｂｙｔｅ）とよびます。

８ビット＝１バイト

１バイトのデータとは例えば次のような形をしています。

第７ビット　　　　　　　　第０ビット

１	０	１	１	０	０	０	１

この各ビットには番号がつけられていて、右から第０ビット、第１ビット、第２ビット、…、第７ビットとよびます。第７ビットを最上位ビット、第０ビットを最下位ビットとよぶこともあります。

１バイトのデータを２進法で表記された数字だと解釈すると、第ｎビットは2^nの重みを持つことになります。上の例で言うと

1	0	1	1	0	0	0	1	
↓		↓	↓				↓	
2^7	+	2^5	$+2^4$	+			2^0	＝　１０進法では 128＋32＋16＋1＝177

となって、この２進数は１０進法では１７７に対応するものになります。

最上位ビットは重みも大きく、ここが１か０かは大きく影響することがあるので、

ＭＳＢ（Ｍｏｓｔ　Ｓｉｇｎｉｆｉｃａｎｔ　Ｂｉｔ）

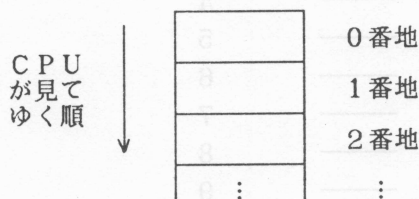
ともよぶことがあります。これに対し、最下位ビットの方は

ＬＳＢ（Ｌｅａｓｔ　Ｓｉｇｎｉｆｉｃａｎｔ　Ｂｉｔ）

ともよばれます。

０－５／メモリー、番地、16進法

ＣＰＵは、メモリーに格納されているプログラムを読み込み、解読することで、所定の動作をする仕組みになっています。

ここでメモリーに眼を転じましょう。メモリーは、ＣＰＵが処理する情報（プログラムやデータ）を格納しておく記憶回路で、情報は１バイト単位で区切られて整然と並べられています。各１バイトの区画には番地（アドレス）とよばれる数字が付されています。番地は、０番地、１番地、２番地…という具合につけられていて、ＣＰＵは、通常、番地の数字の若い方から順に、メモリーを見てプログラム等を読み込み実行していきます。

さて、Ｘ１に搭載されているような８ビット型のＣＰＵでは、番地は１６ビットの数値で表わされるように設計されています。（メモリーの番地もＣＰＵにわかる形――２進数――で表わさなくてはなりませんね。）すなわち次のようになります。

《２進法》		《１０進法》
００００００００００００００００	番地	０番地
０００００００００００００００１	番地	１番地
００００００００００００００１０	番地	２番地
００００００００００００００１１	番地	３番地
⋮		⋮
１１１１１１１１１１１１１１１０	番地	６５５３４番地
１１１１１１１１１１１１１１１１	番地	６５５３５番地

従って、メモリーの番地は、０番地から始まり、最大６５５３５番地までつけることができます。

１バイトのデータにしても、また２バイト（＝１６ビット）で表わされる番地にしても、２進法の０、１の列では私たち人間にはどうもピンと来ませんね。一方、１０進法ではコンピューター側にとって具合が悪い。そこで！と工夫されたのが１６進表記法です。　すなわち、2^4 ＝１６となることを利用し、２進法の数字列を４ビットずつ右から区切って表記していく方法です。各４ビットには次のような記号を対応させます。

《２進法》	《１０進法》	《１６進法》
００００	０	０
０００１	１	１
００１０	２	２
００１１	３	３
０１００	４	４
０１０１	５	５
０１１０	６	６
０１１１	７	７
１０００	８	８
１００１	９	９
１０１０	１０	Ａ
１０１１	１１	Ｂ
１１００	１２	Ｃ
１１０１	１３	Ｄ
１１１０	１４	Ｅ
１１１１	１５	Ｆ

１０進法の１０～１５に相当する部分はもう数字がありませんから、アルファベットのＡ～Ｆをあてることに決められています。

このような１６個の（英）数字を用いると、２進数を忠実に反映し、さらに私たち人間にとってもある程度大きさを想像できるような表記法をすることができます。

図０－４　《１６進表記法の例》

２　進　法	１６進法	１０進法
１０１１　０００１ ４ビットずつ区切る	Ｂ１	１７７
０１０１　１１１１　００１１　０１１１ ４ビットずつ区切る	５Ｆ３７	２４３７５

１６進法と１０進法では共通の数字が多くありますから、混同しないように、１６進法を銘記するには、数のあとにアルファベットのＨをつけることにします。

《例》　５Ｆ３７Ｈ，Ｂ１Ｈ

Ｈは１６進法を意味する英語　Ｈｅｘａｄｅｃｉｍａｌ　の頭文字をとったものです。本書ではこの表記法を採用しますが、他の表記法もあります。

Ｘ１の（Ｈｕ）ＢＡＳＩＣ（正式名称はＣＺ８ＣＢ０１といいます）では、１６進数の頭に＆Ｈをつけています。ＢＡＳＩＣプログラム中で、１６進数を使うにはこの形にしなくてはいけません。

《例》

```
? &H5F37
 24375
Ok
```

上の例のようにキー・インすると、　５Ｆ３７Ｈ　に等しい１０進数　２４３７５　が表示されますね。

また、16進数の頭に$をつける流儀もありますが、この方法は本書では原則として用いません。［注］唯一の例外は付録5のリストです。

16進数と、その表記法について、よろしいですか？　以上まとめておきます。

16進数を表わすのに本書では、 本文中で、数字の後にHをつける BASICプログラム中で、数字の前に&Hをつける という両方法を併用し使い分ける。

さて、1バイトの情報は、16進数2桁で表わされますね（00H～FFH）。また、メモリーの番地を表わす2バイトの数は、16進数4桁で表わされます（0000H～FFFFH）。こうして、0－3節に出てきた英数字の正体がわかります。

```
*M D000
:D000=3E
```

たとえば、モニターを起動してからの上のようなメッセージの意味は、メモリーの　D000H番地に現在格納されている1バイト情報が　3EH　であることを意味しています。他の例についても同様です。

［注］モニターによるメッセージでは、16進数しか扱わないので、Hは省略されています。

0－6／入出力装置

さて、コンピューターの3つの大きな部分のうち、CPU、メモリーについて概説しました。最後の入出力装置について見ておきましょう。

先程のモニター起動画面を思い出して下さい。

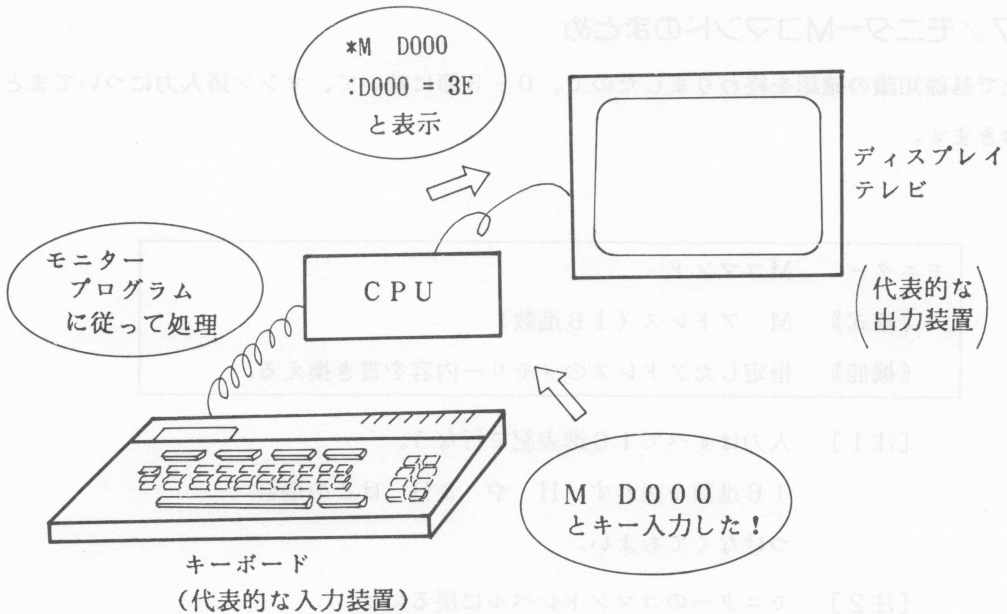
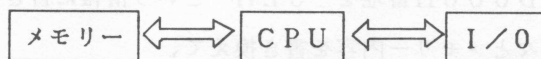



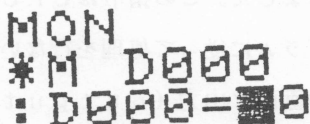
図0－5

パソコンの代表的な入力装置はキーボードです。私たちは、キーボードから M D000 という文字列を入力（ｉｎｐｕｔ）しました。この指令はCPUに伝えられ、ＣＰＵはメモリーに格納されているモニター・プログラムに従って処理を行ないます。この結果は、代表的な出力装置であるディスプレイテレビの画面に出力（ｏｕｔｐｕｔ）され、Ｄ０００Ｈ番地のメモリーの内容が表示された訳ですね。

このように、入力装置や出力装置は、コンピューターが人間と情報のやりとりをする手・目・耳・口のような働きをいたします。これらを総称して、入出力装置(Ｉｎｐｕｔ Ｏｕｔｐｕｔ ｕｎｉｔ）省略して、Ｉ／０などとよびます。

この基本図式をしっかりと頭に入れておいて下さい。

０－７／モニターMコマンドのまとめ

以上で基礎知識の確認を終わりましたので、０－３節に続いて、マシン語入力についてまとめておきます。

モニター	Mコマンド
《書式》	M　アドレス（１６進数）
《機能》	指定したアドレスのメモリー内容を書き換える。

［注１］　入力はすべて１６進表記で行なう。
１６進数を表わす　H　や　&H　はこの場合
つけなくてもよい。

［注２］　モニターのコマンドレベルに戻るには
SHIFT ＋ BREAK による。

たとえば、モニターを起動して、＊M　D０００　とキー・インしたとすると、「メモリーのD０００H番地の内容を書き換える」という指令をCPUに与えたことになります。

```
MON
*M D000
:D000=■0
```

図０－６

すると画面はたとえば上図のようになります。カーソルが点滅している所は、D０００H番地に現在格納されている１バイト情報を１６進数２桁で表示しています。ですから、　３E　とキー・インすることは、D０００H番地を　３EH　という情報に書き換えたことになるのです。０－３節のように次々とメモリー内容を書き換えて、

```
*M D000
:D000=3E
:D001=E3
:D002=01
:D003=F4
:D004=31
:D005=ED
:D006=79
:D007=3E
:D008=03
:D009=01
:D00A=F4
:D00B=21
:D00C=ED
:D00D=79
:D00E=C9
:D00F=00
*
```

図０－７

となるわけですが、これはメモリーの　Ｄ０００Ｈ番地～Ｄ００ＥＨ番地に所定の内容の情報を格納したことになるのです。

では、本当に格納されているかどうかを確認するには、どうすればよいでしょう。それにはモニターのＤコマンドを用います。

０－８／メモリーの内容を見る

メモリー内に記憶されている内容を、ディスプレイテレビ等の出力装置へ出力することをダンプ（ｄｕｍｐ）するといいます。辞書で引くと、「どさりと降ろす」という意味であると出ています。よく、土砂などを運ぶトラックをダンプカーと呼びますね。

さて、ｄｕｍｐの頭文字をとった、モニターのＤコマンドが、メモリー内容のダンプをする命令です。

モニター　Ｄコマンド
《書式》　Ｄ　開始アドレス　終了アドレス
《機能》　指定範囲のメモリー内容を出力装置にダンプする。

早速、試してみましょう。先程私たちが入力した、Ｄ０００Ｈ～Ｄ００ＥＨ番地の内容をダ

ンプしてみます。

```
*D D000 D00E
:D000=3E E3 01 F4 31 ED 79 3E />♥・ホ1▞y>
:D008=03 01 F4 21 ED 79 C9 00 /・・ホ!▞yノ・
*■
```

図0－8

上のように表示されるはずです。上の一列が、D000H～D007H番地の内容を並べたもの、下一列が　D008H～D00FH番地の内容を並べたものです。このように、Dコマンドを実行すると、開始番地から８バイトずつが一列になってダンプされます。

右端の斜線　／　の後に不可思議なマークが８個出ていますね。これは今は気にする必要ありませんが、アスキーダンプと言って、メモリー内容の１６進数に対応するコード（アスキーコード）を持つ文字・記号を表示しています。詳しくは第３章で学びますが、現段階では「メモリーに格納されている意味ある文字列を探すのに用いる」位に理解しておいて下さい。（たとえば　>　のコードが　３ＥＨ　というように読みます）

こうして私たちは、ＢＡＳＩＣで言えば、プログラムを入力する（Ｍコマンド）、リストする（Ｄコマンド）に相当することができるようになったわけですね。最後にプログラムの実行方法について学びましょう。

０－９／マシン語プログラムを実行する

まだ私は「マシン語」とは何かについて正式に説明しておりません。それなのにプログラムの実行なんて！と思われるかもしれませんが、今はともかく「初体験コース」の第０章ですから、理屈はわからなくてもやってしまいましょう！

先程来、Ｄ０００Ｈ番地から　Ｄ００ＥＨ番地へ格納し、ダンプで確認してみたデータ列は、実は、きちんとした意味ある「マシン語プログラム」です。これらの正体については、次章以降、とくに第３章で詳しく学びますが、本節では、ともかく実行してみることにいたします。ＢＡＳＩＣのプログラムでしたら、ＲＵＮなのですが、マシン語の時は？

モニターのＧコマンドを用います。

モニター　Gコマンド

《書式》　G　実行アドレス

《機能》　指定したアドレスから始まるマシン語プログラムを実行する。

準備はよろしいですか？　D000H～D00EH番地には、キチンとデータ（プログラム）が入っていますか？

では　＊G　D000　とキー・インし、CRキーを押して下さい。

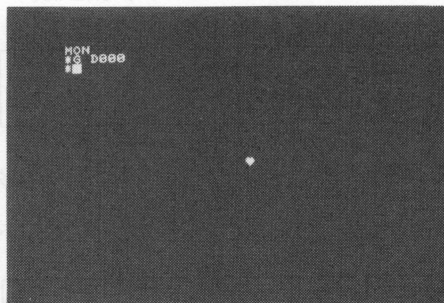



図0－9

正常に動けば（入力ミス等なければ）、画面中央に、赤紫色（マゼンタ）でハートマークが1個表示され、モニターのコマンドレベルに戻るはずです。

そうだったのです。私達が　D000H番地から格納したのは、ハートマーク表示プログラムであったのです。

このプログラムの意味を解明するのは第3章のテーマとなります。また、モニターのGコマンドについても、もう少し注意をする必要がありますが、後まわしにして、本章を終える前に、BASICのコマンドレベルに戻る方法について学んでおきましょう。

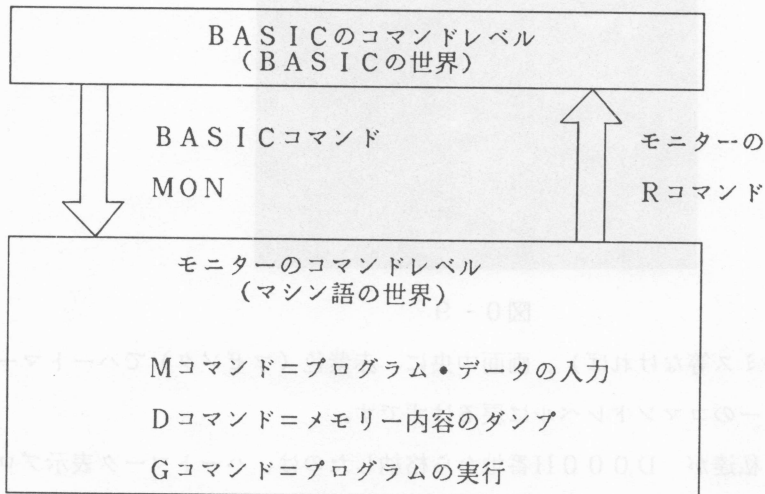
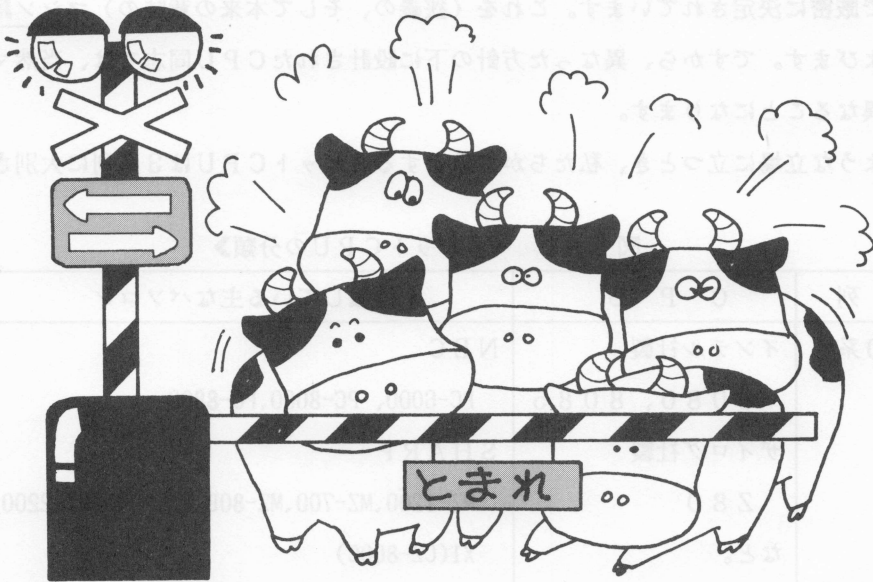
モニター　Rコマンド

《書式》　R

《機能》　モニターのコマンドレベルからBASICのコマンドレベルへ復帰する。

```
*R
Ok
```

実行すると上のようになるはずです。Ｏｋが表示され、ＢＡＳＩＣコマンドレベルに戻ったことがわかります。

以上で私たちは次のようなことを行なえるようになりました。

次章以降、私たちは、ＢＡＳＩＣの世界とマシン語の世界を行きつ戻りつしながら、勉強を進めて行くことになります。これからが、本格的なマシン語の勉強の始まりです。頑張って下さい。

第1章 プログラム作りに挑戦！

■第1章 プログラム作りに挑戦！

1－1／CPUとマシン語

前章において、私たちはＣＰＵの動作原理を確認し、ＣＰＵに理解できるのは、メモリーに格納された０と１の組み合わせ（ビットパターンとよぶ）で表わされる情報だけであることを理解しました。

どのようなビットパターンのとき、ＣＰＵがどのような動作をするかは、ＣＰＵが設計された時点で厳密に決定されています。これを（狭義の、そして本来の意味の）マシン語（機械語）とよびます。ですから、異なった方針の下に設計されたＣＰＵ同志では、当然マシン語の体系は異なることになります。

このような立場に立つとき、私たちが対象とする８ビットＣＰＵは３系列に大別されます。

図１－１　《８ビットＣＰＵの分類》

系　列	Ｃ　Ｐ　Ｕ	搭載している主なパソコン
８０系	インテル社製 ８０８０、８０８５ ザイログ社製 Ｚ８０ など。	ＮＥＣ PC-6000、PC-8000、PC-8800 ＳＨＡＲＰ MZ-1200、MZ-700、MZ-80B、MZ-2000、MZ-2200 X1(CZ-800C) 東　芝　　ＰＡＳＯＰＩＡ，ＰＡＳＯＰＩＡ７ カシオ　　ＦＰ－１１００
６８系	モトローラ社製 6800、6802、6809 など。	富士通　ＦＭ－７、ＦＭ－８ 日　立　　ベーシックマスターシリーズ ナショナル　ＪＲ－１００、ＪＲ－２００
６５０２系	モステクノロジー社製 ６５０２	アップルⅡ

これら３系列で、マシン語の体系は異なり、各々を解説するために１冊ずつ本が必要な程です。

さて、私たちのシャープＸ１は、８０系に属するＺ８０Ａ ＣＰＵ を搭載しています。「Ａ」なんて余計なものがついている、と不安に思われるかもしれませんが、御安心下さい。Ｚ８０ ＣＰＵ と Ｚ８０Ａ ＣＰＵ とはマシン語は全く同じです。何が違うかというと、ＣＰＵを働かせるための水晶発振クロックの周波数が Ｚ８０ では2.5MHz（メガヘルツ）、Ｚ８０Ａ では４MHzとなっていて、Ｚ８０Ａ の方が1.6倍の速さで動作いたします。（Ｚ８０Ａ を Ｚ８０ のＡバージョンとよびます。このような高速版には他に、６MHzのＢバージョンがあります。）

本書ではクロック周波数を考慮しなければならない程微妙なプログラムは扱いませんから、Ｚ８０Ａ も Ｚ８０ も区別せず、Ｚ８０ＣＰＵとして一括して扱うことにいたします。

Ｘ１本体の上ぶたをはずして、中をのぞくと、プリント基板上に整然と並べられたＬＳＩやＩＣ（集積回路＝Ｉｎｔｅｇｒａｔｅｄ Ｃｉｒｃｕｉｔ）たちが見えます。この中に、

ＬＨ００８０Ａ　Ｚ８０Ａ－ＣＰＵ ＳＨＡＲＰ

と書かれているＬＳＩがありますが、これが心臓部の Ｚ８０Ａ ＣＰＵ です。ＬＨ００８０Ａ というのは、シャープが Ｚ８０Ａ ＣＰＵ を製造する時の型番で、Ｚ８０Ａ と全く同じものです。

以上の点よろしいですか？　こうして私たちがこれからＸ１で勉強していくマシン語は、

８０系の Ｚ８０ＣＰＵ のマシン語

であることがはっきりいたしました。

１－２／ニーモニックとアセンブラ

Ｚ８０ＣＰＵは、８ビットＣＰＵですから、そのマシン語も８ビット（＝１バイト）を単位に組み立てられています。これらの本来の姿は、８個のビットパターンですが、私たちはこれと等価な２桁１６進数として扱うことを学びました。

たとえば前章で、私たちが入力し実行してみたマシン語は次のようなものでした。

ﾒﾓﾘｰ ﾉ ｱﾄﾞﾚｽ	ﾏｼﾝｺﾞ	ﾋﾞｯﾄ・ﾊﾟﾀｰﾝ
D000H	3EH	00111110
D001H	E3H	11100011
D002H	01H	00000001
D003H	F4H	11110100
D004H	31H	00110001
D005H	EDH	11101101
D006H	79H	01111001
D007H	3EH	00111110
D008H	03H	00000011
D009H	01H	00000001
D00AH	F4H	11110100
D00BH	21H	00100001
D00CH	EDH	11101101
D00DH	79H	01111001
D00EH	C9H	11001001

図1－2

たしかに、ビットパターンよりは16進表示の方が見やすいのですが、しかし、いくら16進数を眺めていても、これがCPUにどのような動作を指令するマシン語なのかわかりませんね。そこで、各マシン語には、その内容を連想させる暗記用の名前がつけられています。これをニーモニック（mnemonic）とよびます。辞書には「記憶を助ける」と出ています。

暗記用の名前だから各人が勝手につけてもよさそうなものですが、これでは混乱をきたしますので、CPUを開発設計したメーカーにより、ニーモニックのつけ方はきちんと決められています。私たちはザイログ社のZ80CPUを対象としていますので、ザイログ社仕様のニーモニックを用いることになります。

図1－2で例にあげたマシン語には、次のようなニーモニックが対応します。

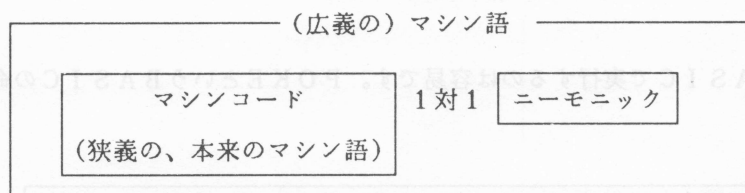
ﾒﾓﾘｰ ﾉ ｱﾄﾞﾚｽ	ﾏｼﾝｺﾞ	
D000H	3EH	----ニーモニック　LD　A，0E3H
D001H	E3H	
D002H	01H	----ニーモニック　LD　BC，31F4H
D003H	F4H	
D004H	31H	
D005H	EDH	----ニーモニック　OUT　（C），A
D006H	79H	
D007H	3EH	----ニーモニック　LD　A，03H
D008H	03H	
D009H	01H	----ニーモニック　LD　BC，21F4H
D00AH	F4H	
D00BH	21H	
D00CH	EDH	----ニーモニック　OUT　（C），A
D00DH	79H	
D00EH	C9H	----ニーモニック　RET

図1－3

マシン語は、１バイトを単位として組み立てられていると述べましたが、１つのまとまった意味をもつマシン語は、１バイト～４バイトで表わされています。その各々に、１つのニーモニックが対応し、上図はその様子を示しています。たとえば、２バイト命令　３Ｅ　Ｅ３　は、ニーモニック　ＬＤ　Ａ，０Ｅ３Ｈ　で表わされ、３バイト命令　０１　Ｆ４　３１　は、ＬＤ　ＢＣ，３１Ｆ４Ｈ　で表わされ、１バイト命令　Ｃ９　は、ニーモニック　ＲＥＴ　で表わされるというようになっています。

ニーモニックは処理内容を表わす英単語ないしはその省略形からできています。上の例でいうと、ＬＤ　は　ｌｏａｄ　の略、ＲＥＴ　は　ｒｅｔｕｒｎ　の省略形です。

このように、マシン語の１６進コード（１～４バイト）は、ニーモニックと１対１に対応するようになっていますから、しばしばニーモニック表記の方も（広義の）マシン語とよぶことがあります。本書では、マシン語をなるべく広義の意味に用います。そして、１６進コードの方をマシンコードとよぶことにします。

《本書での用語の使い方》

人間にとっては、ニーモニック表記でマシン語プログラムを組んでいった方が、「何をしているのか」がわかり、便利なのですが、ＬＤ　Ａ，０Ｅ３Ｈ　などと入力してもＣＰＵには理解できません。ニーモニック表記されたプログラムを、ＣＰＵに理解できるマシンコードに変換する作業をアセンブル（ａｓｓｅｍｂｌｅ）とよびます。人間の手で、この作業を行なう場合、ハンドアセンブルといいます。

私たちは本書で、ハンドアセンブルの練習を行なうことになりますが、少し長いプログラムになると大変な作業になります。そこでアセンブル作業をコンピューターに行なわせることを考えます。このための自動変換プログラムをアセンブラ（ａｓｓｅｍｂｌｅｒ）とよんでいます。ニーモニック（マシンコードに変換される）と、マシンコードには変換されないけれどもアセンブラに指示を与える命令（疑似命令という）とからなる体系をアセンブリ言語（ａｓｓｅｍｂｌｙ　ｌａｎｇｕａｇｅ）とよびます。本来は、アセンブリ言語をきちんと学んだ上

で、実際にアセンブラを動かして、マシン語プログラムを作成していくのが望ましい姿でしょうが、現時点でＸ１においてはまだ満足できるアセンブラがないようです（製品として発売されていないという意味で）。そこで、しかたなく本書では、タップリとハンドアセンブルに挑戦していただくことになります。でも頑張って下さい。私もそのように勉強してきたのですから。Ｘ１に便利なアセンブラができて、ハンドアセンブルから解放される日が来るのを夢みて、せっせとマシン語の実力を蓄えていきましょう！

１－３／Aレジスタ登場！

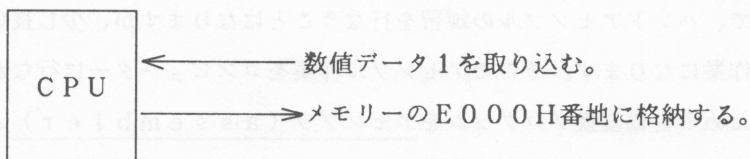
では、いよいよ「正式に」マシン語を勉強していくことにします。用意はよろしいですか。

まず、最初に学ぶマシン語は、ＣＰＵとメモリーの間のデータのやりとりに関するものです。

課題１　メモリーのＥ０００Ｈ番地に数値データ１を格納すること。

同じことをＢＡＳＩＣで実行するのは容易です。ＰＯＫＥというＢＡＳＩＣの命令を用いればよいのです。

課題１に対する解答（ＢＡＳＩＣ版） ＰＯＫＥ　&ＨＥ０００，１

このような簡単かつ基本的な処理は、マシン語ではどのように行なわれているのでしょうか？　実は、ＣＰＵは、２段構えで、この処理を行ないます。

このことを正確に表現するには、レジスタという言葉を覚えなくてはなりません。

私は現在、この本の執筆をするため原稿用紙に向かっています。本棚には、コンピューターの本がずらり（？）と並んでいます。本棚はメモリーに相当すると思って下さい。執筆という情報処理を行なう私は、いわばＣＰＵです。さて、私は執筆に必要な資料を参照するために本棚（メモリー）から取り出し、仕事場である机のまわりの手の届く所に一時的に置きます。このような場所にあたるのがレジスタです。自分の手許にデータを置いておけば、本棚（メモリー）に取りに行くより速く処理することができます。一方、そのような場所にあるデータは紛失したり散乱したりするかもしれません。あくまで長期的に保存するには本棚（メモリー）にきちんと整理して格納しておく必要があるのです。

以上、例え話をつかって説明してみましたが、きちんというと次のようになります。

レジスタ（ｒｅｇｉｓｔｅｒ） メモリーと同様に記憶用の回路であるが、ＣＰＵ内部にあって、データを一時的に記憶しておくために用いられる。

Ｚ８０ＣＰＵ内には、たくさんのレジスタがあります。これらの詳細は次章で学ぶことにして、本章では、Ａレジスタについてだけ学ぶことにします。

Ａレジスタ アキュムレータ（ａｃｃｕｍｕｌａｔｏｒ＝累算器） ともよばれる８ビットレジスタで、ＣＰＵが処理するさまざまな算術・論理演算において中心的な役割をはたす。

さて、課題１に戻りましょう。処理手順は正しくは次のようになります。

Ａレジスタに数値データ１を取り込む。
Ａレジスタの内容（今の場合は１）を、メモリーのE000H番地へ格納する。

もう私たちはマシン語の世界へ片足をつっこみました。上の各処理は、ＣＰＵが行なう基本的な処理そのものなのです。ニーモニックでは、次のように表記されます。

ＬＤ　Ａ，０１Ｈ
ＬＤ　（０Ｅ０００Ｈ），Ａ

この表記と、処理内容を考え合わせると、表記の意味は何となくわかりますね。次節で、この問題をきちんと学びましょう。

１－４／ニーモニック表記のルール

前節で登場した２つのニーモニックについて考えてみましょう。まず両者共通の　ＬＤ　という部分を取り上げます。

ＬＤ　は、　ｌｏａｄ　の最初と末尾をとった省略形です。ＢＡＳＩＣ命令にも　ＬＯＡＤ　というのがあります。「テープからプログラムをロードする」などと使うように、テープやフロッピーディスク等に格納されているプログラム・データをメモリーへ転送することを指しています。　ｌｏａｄ　という英単語の原義は「積み込む」という意味ですが、コンピューターではこのように「転送する」という意味あいでも用いられます。（この他に、ｔｒａｎｓｆｅｒというのもあります）

従って、　ＬＤ　のつく命令は、データをある場所からある場所へ転送するもので、一般に転送命令とよばれています。

転送命令では、ちょうど郵便配達を想像するとわかるように、差し出し側と届け先が存在しますね。コンピューター用語では、差し出し側をソース（ｓｏｕｒｃｅ＝源の意）、届け先をデスティネーション（ｄｅｓｔｉｎａｔｉｏｎ＝目的地の意）とよんでいます。

さて、ザイログ社仕様のニーモニックでは次のようなルールがあります。

《ルール１》

転送命令では、デスティネーション、ソースの順に書く。

今の例でいうと

	デスティネーション		ソース
ＬＤ	Ａ	，	０１Ｈ
ＬＤ	（０Ｅ０００Ｈ）	，	Ａ

となっています。横書きの時に私たちは左から右へ書き進めますが、データ転送表記の向きは逆であることに、まず注目して下さい。

さて、次のルールです。（０Ｅ０００Ｈ） に関するものです。まず、 ０Ｅ０００Ｈ の頭の ０ はどんな意味でしょうか？

《ルール２》

> １６進表記で数を表わすとき、アルファベットのＡ～Ｆで始まる数は、その頭に ０ をつける。

これは次章ではっきりとすることですが、レジスタにはＡレジスタの他に、Ｂ～Ｆという名前を冠したレジスタもあり、Ａ～Ｆがレジスタを指すのか、１６進数なのか混同しないように区別するためのルールです。

また本書の本文でも採用しているルールですが、

《ルール３》

> １６進数は末尾に Ｈ をつけて表わす。

ニーモニック表記では特にきちんと守る必要があります。

最後に （０Ｅ０００Ｈ） の （ ） について見ておきましょう。

《ルール４》

> （ ）を用いると、括弧の中味で指定されたメモリー等の番地を表わす。

これらのルールを知れば、もうニーモニック表記の意味はおわかりですね。

ＬＤ　Ａ，０１Ｈ	＝Ａレジスタへ０１Ｈをロードする。
ＬＤ　（０Ｅ０００Ｈ），Ａ	＝メモリーのＥ０００Ｈ番地へ Ａレジスタにあるデータをロードする。

転送命令で１つ確認しておくことがあります。たとえば、　ＬＤ　（０Ｅ０００Ｈ），Ａ　を実行すると、Ａレジスタの中味は空になると思いがちですが、そうではないのです。ＢＡＳＩＣでも、

Ａ＝１
Ｍ＝Ａ

を実行すると、変数Ａの中味１は残り、新しく変数ＭにＡの内容がコピーされて、結局ＭもＡも中味は１になりますね。これと全く同様です。転送命令では、ソースの中味は不変であるという原則を、よく覚えておいて下さい。

［注］　ＬＤ　Ａ，０１Ｈ　の　０１Ｈ　の部分に疑問を持たれた読者もおられると思います。０１Ｈ　の頭の　０　は、ルール２で言うところの　０　ではありません。たいていのアセンブラでは、　ＬＤ　Ａ，１Ｈ　と書いても、　ＬＤ　Ａ，１　と　１　を１０進表記しても受けつけてくれるはずです。しかし慣れないうちは、　１Ｈ　とか　１　と書くと何ビットデータなのか間違えることがあります。Ａレジスタは、８ビットのレジスタですから、数の１はあくまで

ビット・パターン	１６進表記
００００００００１	０１Ｈ

として格納されるわけです。従って、慣れるまではなるべく、８ビット（後出する１６ビットのときも）のデータで、上位の４ビットが０のときは、　０　を省略せずに表記した方がよいと思います。本書の本文では、この方針で一貫します。

１－５／ハンドアセンブルの注意点

ニーモニック表記の意味がわかると、次はこれをＣＰＵに理解できるマシンコードに変換する作業をしなくてはなりません。私たちは、ハンドアセンブルにより行ないましょう。

そのためには、マシンコードに翻訳する辞書が必要です。付録の「Ｚ８０命令表」がこの役割を果たしてくれます。8ビットロード命令の所を見て下さい。

図１－４　《8ビットロード命令》

	×	A	B	…………	n	I	R
ＬＤ	Ａ，×	7F	78	…………	3E n	ED 57	ED 5F
	⋮	⋮	⋮	…………		---	---

×印で書かれた最上段の１行は、ソースを表わしています。今の場合、ソースは8ビット数値データですから、　n　と記されている所を見ます。

さて、次の２行目は、デスティネーションがAレジスタであるロード命令を集めてあります。この行で、　n　の列を見ると、

３Ｅ n

と書かれています。これは、　ＬＤ　Ａ，ｎ　のマシンコードが２バイトであって、　３Ｅ　n　であることを意味しています。私たちの場合、　n　は　０１Ｈ　ですから、　ＬＤ　Ａ，０１Ｈ　のマシンコードは　３Ｅ　０１　となるわけです。同様に、Aレジスタに数値ＦＦＨ（＝１０進法で２５５）　をロードするなら、

ニーモニック	マシンコード
ＬＤ　Ａ，０ＦＦＨ	３Ｅ　ＦＦ

となるわけです。

ＬＤ　Ａ，ｎ　の機能は、Ａレジスタに８ビット数値をロードするものですが、以後、言葉で説明するかわりに、　Ａ←ｎ　と矢印を用いて表わすと便利ですね。このようにして、私たちが初めて学んだマシン語の知識は次のようにまとめられます。

解　説

ニーモニック：ＬＤ　Ａ，ｎ
　　　　　　（ｎは８ビット数値）
マシンコード：３Ｅ　ｎ
機　　　　能：Ａ　←　ｎ

次は、ＬＤ（０Ｅ０００Ｈ），Ａ　の番ですね。これをマシンコードに直しましょう。再び「Ｚ８０命令表」で８ビットロード命令の項を見ます。ＬＤ（ｎｎ′），×　と書かれた行を注目して下さい。

図１－５　《８ビットロード命令》

×	A	B	…………	R
LD （ｎｎ′），×	32 n′ n		…………	

今の場合、デスティネーションは指定された番地のメモリーです。　ｎｎ′　は　ｎ　を上位８ビット、　ｎ′　を下位８ビットとする１６ビット数値を表わし、　（ｎｎ′）　は、メモリーの　ｎｎ′番地を意味します。この行を横に見ると、１か所を除き、すべて空欄です。すなわち、デスティネーションが　（ｎｎ′）　のときは、ソースはAレジスタに限られてしまいます。

こうして、　ＬＤ　（ｎｎ′），Ａ　というニーモニックに対応するマシンコードは

３２
ｎ′
ｎ

すなわち、　３２　ｎ′　ｎ　ということになります。

ちょっと変だと思われた方がおられると思います。　ｎ′　と　ｎ　の書きちがいでは？という疑問かと思いますが、これはミスプリではなく、きちんとしたルールなのです。

《上下位逆転の原則》

> Ｚ８０をはじめ、一般に８０系のＣＰＵでは、２バイトの数をアセンブルするとき、その上位１バイト、下位１バイトを逆転させる。

慣れないうちは奇妙な原則に思えるかもしれませんが、８０系ＣＰＵの設計方針（アーキテクチュア）からするとそれなりの合理性があるようです。

話を戻します。メモリーの番地を指定する　ｎｎ′　は２バイト数値ですから、上下位逆転の原則が適用され、マシンコードには、下位１バイトの　ｎ′　が先に、上位１バイトの　ｎ　が後になって変換されるのです。こうして、　ＬＤ　（ｎｎ′），Ａ　のマシンコードが、

３２　ｎ′　ｎ　となる訳です。私たちの場合、ｎｎ′＝Ｅ０００Ｈで、ｎ＝Ｅ０、ｎ′＝０

０ですから、

ニーモニック	マシンコード
ＬＤ　（０Ｅ０００Ｈ），Ａ	３２　００　Ｅ０

と変換されます。この上下位逆転則は、いろいろな場面で以後登場してきますから、覚えておいて下さい。

さあ、私たちが２番目に学んだマシン語についてまとめておきましょう。

解　説

ニーモニック：ＬＤ　（ｎｎ′），Ａ
　　　　（ｎｎ′は１６ビット数値）
マシンコード：　３２　ｎ′　ｎ
機　　　能：（ｎｎ′）　←　Ａ

１－６／暴走の恐怖！

前節において私たちは初めて、ハンドアセンブルを経験いたしました。マシンコードにしてしまえば、それはＣＰＵに理解できる形ですから、プログラムとしてメモリーに格納し実行してみたいですね。

ニーモニック	マシンコード
ＬＤ　Ａ，０１Ｈ	３Ｅ　０１
ＬＤ　（０Ｅ０００Ｈ），Ａ	３２　００　Ｅ０

たとえば、メモリーのＤ０００Ｈ番地から、このプログラムを格納してみましょう。前章で学んだように、モニターを起動し、次のように入力して下さい。

図１－６

メモリーの何番地に、どういうデータを入れるか、よろしいですね。しかし、もう少しニーモニックと対応させるとわかりやすくなります。

X１のモニターは優れた画面編集機能（スクリーン・エディット）を持っていて、同様な結果は次のように入力しても得られます。

```
MON
*M D000
:D000=3E01
:D002=3200E0
:D005=00
*■
```

図１－７

どうですか？　モニターが自動的に番地を進めてくれますね。このように、ニーモニックと対応するように入力していくと、誤入力を少なくすることができます。

ｱﾄﾞﾚｽ	ﾏｼﾝｺｰﾄﾞ	ﾆｰﾓﾆｯｸ
D000H	3E01	LD　A,01H
D002H	3200E0	LD　(0E000H),A

これから上のようなリストが続々と出て来ますから、是非慣れていただきたいと思います。

現在作成中のマシン語プログラムに対応することをＢＡＳＩＣで書くと次のようになります（Ａレジスタを変数Ａと見たてています）。

```
10 A=1
20 POKE &HE000,A
```

このＢＡＳＩＣプログラムはＲＵＮすれば、もち論正常に動きます。結果を知りたければ、次のようにすればよいのでしたね。（メモリーの内容を見るには、ＰＥＥＫ関数を用います！）

```
? PEEK(&HE000)
 1
OK
```

たしかに、メモリーの　Ｅ０００Ｈ番地には、数値１が格納されています。

ですから、私たちのマシン語プログラムも即実行と行きたい所ですが、１つ重大なことを忘れているのです。

モニターのＧコマンドで、＊Ｇ　Ｄ０００　として、私たちのマシン語プログラムを実行すると、ＣＰＵは、メモリーの　Ｄ０００Ｈ番地に注目し、次のような処理を続けていくことになります。

《ＣＰＵの動作》

① メモリーの　Ｄ０００Ｈ番地にある　３ＥＨ　をとり込み解読する。
② これが　ＬＤ　Ａ，ｎ　型の２バイト命令であることを知り、ｎ　を探して次の　Ｄ００１Ｈ番地にある　０１Ｈ　をとり込む。
③ Ａレジスタに　０１Ｈ　をロードする。
④ 次の命令を解釈するため、Ｄ００２Ｈ番地にある　３２Ｈ　をとり込み解読する。
⑤ これが　ＬＤ　（ｎｎ′），Ａ　型の３バイト命令であることを知り、ｎｎ′　を探して、次の　Ｄ００３Ｈ番地と　Ｄ００４Ｈ番地の内容をとり込む。
⑥ メモリーの　Ｅ０００Ｈ番地へＡレジスタの内容をロードする。
⑦ 次の命令を探しに、Ｄ００５Ｈ番地に注目する。

⋮

ここで⑦が重大です。ＣＰＵは融通をきかせてくれませんから、プログラムの停止を指示されるまで、このような動作を際限なく繰り返していくことになります。多くの読者はＢＡＳＩＣのシステムテープをロードした後に、上の実験をしているでしょうから、メモリー内がクリアされていて異常は起きないと思います。しかし、ゲーム等で遊んでメモリー内に沢山の残渣を残した後に、上のことをしたらどうなるでしょう。一応、Ｄ００５Ｈ番地以降のメモリー内には、私たちにとって未知のデータが格納されていると思わなくてはなりませんね。それでも、ＣＰＵはまだプログラムが続いていると認識し、実行していきますから、場合によっては予想もしないことが起こるかもしれません。このような事態を「暴走」とよんでいます。

危険なことですが、わざと暴走を起こさせてみます。Ｄ００５Ｈ番地以降の「未知のデータ」として、あらかじめ次のように設定してみて下さい。

図1－8　《暴走の実験1》

```
MON
*M D005
:D005=C3
:D006=00
:D007=D0
:D008=00
*R
Ok
```

よろしいですか？　今、入力したものは、「私たちにとって未知のデータ」であるとしますよ！　さあ、こうして、D000H番地からのプログラムを実行してみて下さい。いかがですか？　カーソルが消えて、いくらキーを押しても、 SHIFT ＋ BREAK さえも効かない状態になってしまいました。これは「未知のデータ」により、プログラムが無限ループに入ってしまったことによる暴走です。

今の場合は、私がわざと設定したいたずらですから、悲劇的な結果にはなりません。マシン語プログラムの暴走は、あくまで「未知な原因」により起こるもので予断はできないのですが、幸いにして軽度な暴走なら直せる場合があります。X1の背面に、リセットボタンがありますね。これを押して下さい。実験1のような軽症の暴走は大抵これで解除され、画面に　Ok　が表示されて、BASICのコマンドレベルに戻るはずです。

しかし、いつもこうだとは思わないで下さい。次に取り返しのつかない暴走例を体験していただきます。今度の「未知のデータ」として、次のように設定してみて下さい。

図1－9　《暴走の実験2》

「未知のデータの設定」完了しましたか？　今度は確実に悲惨な結果になること請け合いますので、プログラム等の壊したくないデータがありましたら、今のうちにテープ等にセーブしておいて下さい。「悲惨な暴走実験」の用意よろしいですか？

では深呼吸をして、モニターを起動し、＊G　D000　で私たちのプログラムを実行して下さい。いかがですか？　メチャクチャなことが起こったでしょう！　今回の暴走は、リセットボタンを押しても、ＩＰＬが起動するだけです。こうして私たちは、ＢＡＳＩＣのシステムを破壊し、ＩＰＬを呼び出してしまったのです。メモリー内の全プログラム、データはもう取り戻すことはできません。

「暴走」の恐ろしさがおわかりになりましたか？　こうして、私たちは「痛い代償」を払って、大切なことを学んだのです。

《教訓》

(1)　マシン語プログラムの最後には、何らかの実行停止措置を講じなければならない。
(2)　マシン語プログラムを自作する時は、十分なデバッグ（ミスの取り除き）を行なう必要がある。
(3)　自作のマシン語プログラムを実験的に走らせる前に、なるべくセーブをしておいた方がよい。

この教訓に基づき次節において、私たちはマシン語でのプログラムの停止法を学ばなくてはなりません。

［注］　ＢＡＳＩＣプログラムでは、最後にプログラム終了宣言である　ＥＮＤ　を置かなくても、多くの場合、システムが処理をしてくれて、プログラムは止まるようになっています。しかし、すぐ後に、サブルーチンが続いている時などは、　ＥＮＤ　を置かないと、プログラムは止まらずに、サブルーチンへ飛び込み、　ＲＥＴＵＲＮ　ｗｉｔｈｏｕｔ　ＧＯＳＵＢ　エラーを生じたりします。ですから、ＢＡＳＩＣプログラムのときも、なるべくＥＮＤ　を書くようにした方がよいのです！

１－７／プログラムの止め方をマスター！

Ｘ１でマシン語を扱う時のプログラムを止める理屈は、実は初心者の方には少し難しい部類に属します。まず、ズバリ答えだけ言ってしまいますと、次のようになります。

> Ｇコマンドにより、マシン語プログラムを走らせる時は、あらかじめプログラムの最後にマシンコード　Ｃ９　を置いておかねばならない。このようにしておくと、モニターのコマンドレベルに戻り、プログラムは停止する。

まず理屈抜きで実験いたします。次の１バイトプログラムを入力し、実行して下さい。

```
MON
*M D000
:D000=C9
:D001=00
*■
```

図１－１０

＊Ｇ　Ｄ０００　により実行すると、すぐに　＊印を表示しカーソルが点滅して、モニターのコマンド待ちに戻るはずです。

もう１つ実験をいたします。１－３節以来の懸案であった「Ｅ０００Ｈ番地に１を格納するプログラム」ですが、末尾に　Ｃ９　を加えて、次の形で入力してみましょう。

```
MON
*M D000
:D000=3E01
:D002=3200E0
:D005=C9
:D006=00
*■
```

図１－１１

前節では「暴走」を体験していただいたので、コワゴワでしょうが、今回は安全なことを保証いたしますから、＊Ｇ　Ｄ０００　で実行して下さい。

いかがですか？　前の実験と同じく、モニターのコマンド待ちに戻りましたね。つまりプロ

グラムを止めることに成功したのです！

結果を見ましょう。　Ｅ０００Ｈ番地　に　０１Ｈ　が格納されていることを見るにはどうすればよいのでしたか？　そう、Ｄコマンドでダンプすればよいのですね。さっそく、＊Ｄ　Ｅ０００　により　Ｅ０００Ｈ番地から、メモリー内容をダンプしてみましょう。

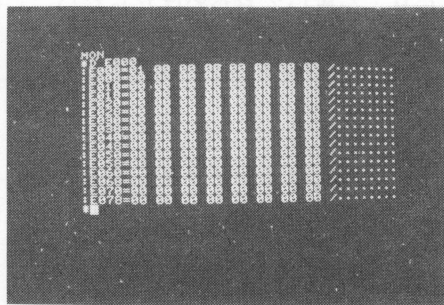



図１－１２

いかがですか？　ちゃんと、Ｅ０００Ｈ番地に　０１Ｈ　が格納されていますね。つまり、私たちは、１－３節の課題に対する解答を得たことになるのです。

では、マシンコード　Ｃ９　で表わされるマシン語は、どのような内容のものなのでしょうか？　本節冒頭で述べたように、実はこの理屈が難しいのです。マシン語サブルーチンの実行の原理とかかわりますので、詳細は第３章に任せて、本節では簡単に説明しておきます。

マニュアル１７６ページにモニターのＧコマンドについて説明があります。「ゴーサブ」と出ていますね。これがキーポイントです。

ＢＡＳＩＣでは、ある番地へジャンプさせる命令として　ＧＯＴＯ　と　ＧＯＳＵＢ　があることは御存じと思います。マニュアルにあるＧコマンドの説明は、Ｇコマンドによるマシン語プログラムの実行方法が、ＢＡＳＩＣでの　ＧＯＳＵＢ　に相当する形で行なわれるということを言っています。

ＢＡＳＩＣでは、ＧＯＳＵＢ文はサブルーチンの実行を指令する命令でした。そして、サブルーチンの終わりには必ず　ＲＥＴＵＲＮ　が必要でしたね。マシン語においても同様で、Ｇコマンドでマシン語プログラムを実行すると、そのプログラムはモニターのコマンドレベルから呼び出されるサブルーチンと見なされますので、末尾にはリターン命令が必要なのです。ＢＡＳＩＣで　ＲＥＴＵＲＮ　を実行すると、サブルーチンを呼び出した次の行に復帰するよう

に、マシン語においてもリターン命令を実行すると、サブルーチンを呼び出したモニターのコマンドレベルに復帰することができます。＊印が表示されたのは、この理由によります。

一応、知識としてまとめておきましょう。

解　説

ニーモニック：　ＲＥＴ

マシンコード：　Ｃ９

機　　　　能：　マシン語サブルーチンからの復帰

いかがですか？　この説明だけでは、きっとおわかりにならない方もおられると思います。それは当然のことで、真の理解に到達するには、マシン語におけるサブルーチンの実行原理が説明されなくてはなりません。これに関しては、第３章の３－１７節以降でタップリと説明を加えますから、それまでお待ち下さい。わからない方は、現段階では、次のように理解しておいて下さい。

解　説

ニーモニック：　ＲＥＴ

マシンコード：　Ｃ９

機　　　　能：　モニターのコマンドレベルに戻ることで、
　　　　　　　　プログラムを停止させるおまじない。

よろしいですか？　従いまして、１－３節の課題に対する解答は、次のようになります。

<u>課題１に対する解答</u>（マシン語版）

```
アドレス    マシンコード     ニーモニック
D000H      3E01           LD    A,01H
D002H      3200E0         LD    (0E000H),A
D005H      C9             RET
```

１－８／止め方についての注意

本節の内容は、少し細かい所に立ち入りますので、初心者の方は読みとばして差しつかえありません。

ＰＣ－８００１等でマシン語を勉強されたことのある方は、プログラムの止め方として次のものがあることを御存じでしょう。

①　モニターのホットスタートへジャンプする方法。

②　ＣＰＵを停止させる方法。

①の場合は、Ｘ１の（Ｈｕ）ＢＡＳＩＣでのモニターホットスタート番地は　１０００Ｈ番地なので、　ＪＰ　１０００Ｈ　によりここへジャンプさせれば、モニターのコマンドレベルに戻ってプログラムは停止します。もち論この止め方でも構いません（第３章３－２１節参照）。

次に②の場合ですが、マシンコード　７６Ｈ　のＨＡＬＴ命令を実行すれば、ＣＰＵは停止（ＨＡＬＴ）状態となり、原理的にはプログラムが停止するはずです。しかし、Ｘ１の場合は、この方法ではプログラムが止まらなかったり、止まってもリセットボタンでＢＡＳＩＣに復帰するとシステムが壊れていたりすることがあります。この原因は複雑でしょうが、おおよそ次のためと思われます。

ＨＡＬＴ状態でも、ＣＰＵの機能は完全に停止しているのではなく、割り込みを受けつけることができます。ところで、Ｘ１においてはそのハードウェアの構造上、キー入力をサブＣＰＵからの割り込みにより処理しています。このために、メインＣＰＵであるＺ８０には、頻繁に割り込みがかかることになり、ＨＡＬＴ状態が解除されてしまうのです。

従って、Ｘ１でマシン語プログラムを実行するには、　ＨＡＬＴ　により停止する方法は適切ではありません。注意して下さい。

本書においては、マシン語初心者の方々を読者として想定しておりますので、当面、モニターの内部構造を利用した止め方　ＪＰ　１０００Ｈ　を利用することはせず、マニュアルにあるような　ＲＥＴ　による止め方を採用いたします。

1－9／BASICインタプリタについて

私たちは、初めて自らの手でマシン語プログラム（短いものですが）を作成し、実行することに成功いたしました。本章を終えるにあたり、今まで曖昧に残してきた「ＢＡＳＩＣのシステムプログラム」というものについて少し考えておきましょう。

まず、モニターを起動し、＊Ｄ　００００　９ＦＣ４　を実行して下さい。すさまじい速さでダンプリストが画面を下から上に流れていきます（スクロールという）。すべてダンプし終わるのに約３分３０秒かかりますが、画面を注目していてわかることは、これらはすべてマシン語であるということです。これが、私たちの日頃親しんでいる「ＢＡＳＩＣ言語」の真の姿なのです。

私たちは、前章と本章でＣＰＵが理解できるのはマシン語だけであることを知りました。従って、　ＰＲＩＮＴ“Ａ”　などという文字列は当然ＣＰＵには理解不能ですね。ちょうどニーモニック表記をアセンブルしてマシン語に直したように、私たちにとって理解しやすい　ＰＲＩＮＴ“Ａ”　などの命令も、マシン語に翻訳してやる必要があります。

アセンブラもこのような翻訳プログラムの１つですが、アセンブリ言語の水準はＣＰＵの動作と密着していて、人間がプログラムを作成するには大変な手間がかかります。もっと人間の水準に近づけて、人間が日常使用する言語に近い言葉をマシン語に翻訳できればプログラムはずっと作りやすくなります。このような翻訳プログラムを高級言語とか高水準言語とかよびます。　BASIC　もその１つですし、他に　FORTRAN，　COBOL，　PASCAL　などの言語もそうです。これらの翻訳プログラム自体は、ＣＰＵに理解できなくてはなりませんから、先程、ＢＡＳＩＣの真の姿を見たようにマシン語で書かれています。

さて翻訳の方法には２種類あります。大型コンピューター等で常識的に採用されている方法は、日常言語に近い言語で書かれたプログラムを一挙にマシン語に変換してしまうものです。これをコンパイラとよびます。ＦＯＲＴＲＡＮ，ＣＯＢＯＬ，ＰＡＳＣＡＬ等の言語は普通コンパイラ方式をとっています。

もう１つの方法はプログラムを一挙にマシン語化するのではなく、少しずつ翻訳しては、その内容にあたる処理を行なうマシン語サブルーチンを呼び出していく方法で、インタプリタとよばれます。インタプリタ方式の代表は　ＢＡＳＩＣ　です（最近は　ＬＯＧＯ　という言語もありますね）。

コンパイラでは、一度マシン語化してしまえば、実行時には完全なマシン語プログラムとな

っていますから高速処理を行なうことができます。一方、インタプリタでは、逐語訳的に翻訳しては実行というパターンを繰り返すために、どうしてもある限度以上のスピードは出せません。よく、ＢＡＳＩＣでゲームを作るとリアルタイム（プログラムでの時間の進行と現実でのそれが同じであること）処理には不向きだと言われますが、それは、パソコンのＢＡＳＩＣがインタプリタであるからなのです。本書で、私たちはマシン語を利用することで、高速ゲーム作りに挑戦いたします。

さて、Ｘ１の基本的なシステムプログラムであるＢＡＳＩＣインタプリタも巨大なマシン語プログラムであり、実に全メモリーの３分の２近くを占領しています。第０章０－２節で私たちの周囲にウンザリする位マシン語があると述べたのは、こういうわけなのです。

Ｘ１では、他の言語も使えるようにメモリーをできるだけ自由な形にしてあります（クリーン設計という）。メモリーには、読み書き自由のＲＡＭ（Ｒａｎｄｏｍ　Ａｃｃｅｓｓ　Ｍｅｍｏｒｙ）と、読みとり専用のＲＯＭ（Ｒｅａｄ　Ｏｎｌｙ　Ｍｅｍｏｒｙ）の２種がありますが、Ｘ１は６５５３６バイト分のメインメモリーをすべてＲＡＭにしてあります。よく　６４Ｋ　フルＲＡＭ　などと言っていますね。コンピューターでは、　2^{10}＝１０２４　のことを　Ｋ　という単位で表わします（キロと区別するためケーと読むそうです）。　６５５３６バイト＝６４Ｋバイト　となるわけですね。

しかし、これでは電源投入時にメモリー内は空で何もできませんから、テープからシステムプログラムを読むための最小限のプログラムはＲＯＭに入れて消えないようにしてあります。これをＩＰＬ（＝Ｉｎｉｔｉａｌ　Ｐｒｏｇｒａｍ　Ｌｏａｄｅｒ）とよんでいます。電源投入時に

ＩＰＬ　ｉｓ　ｌｏｏｋｉｎｇ　ｆｏｒ　ａ　ｐｒｏｇｒａｍ ……

とメッセージが出ますが、あれはＩＰＬが動作しているのです。ＩＰＬＲＯＭはシステムプログラム（ＢＡＳＩＣインタプリタ等）を読み終わると、役目を終えてメインメモリーから切り離され、メインメモリーはすべてＲＡＭになります。

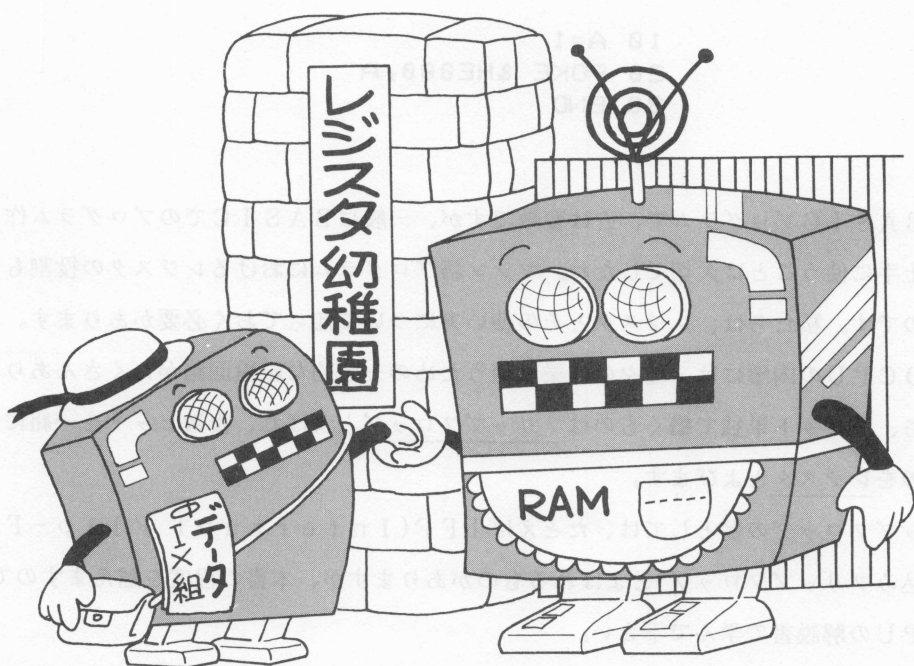
というわけで、Ｘ１においてはＢＡＳＩＣインタプリタもＲＡＭ上にあり、私たちはこれを書き換えてしまうことが可能です。しかしそのためにはＢＡＳＩＣインタプリタの内部を熟知してからでないと、「暴走」の危険がありますね。マニュアルの６１ページにある注意はそのためのものです。

ＰＯＫＥの使用に際しては、ＢＡＳＩＣプログラムなどメインメモリー内のシステムを破壊してしまう恐れがあるので十分注意して下さい。

ＢＡＳＩＣインタプリタのシステムプログラムは、メモリー上　０００００Ｈ～９ＦＣ４Ｈ番地に置かれています。また、私たちが行番号つきで入力するＢＡＳＩＣプログラムは、９ＦＣ５Ｈ番地以降に格納されます。

前節まで、私たちは、マシン語プログラムをＤ０００Ｈ番地から格納しましたが、これはシステムやＢＡＳＩＣプログラムを壊さぬようにする配慮のためです。第３章において、私たちは、「マシン語プログラムをどこに置くか」に関して、もっとはっきりしたことを学ぶでしょう。

本章では、Ａレジスタとメモリーの情報転送を学びましたが、次章からは、もう少し多彩なプログラムに挑戦いたします。御期待下さい。

第2章 レジスタに挑戦！

■第2章 レジスタに挑戦！

2－1／全レジスタそろいぶみ！

前章において、私たちはAレジスタ（アキュムレータ）について学び、初めてマシン語プログラムの作成を体験いたしました。このプログラムはＢＡＳＩＣでの次のものに相当していました。

```
10 A=1
20 POKE &HE000,A
30 END
```

このＢＡＳＩＣプログラムで、Aは変数ですが、一般にＢＡＳＩＣでのプログラム作りでは、変数を上手に使うことは大切でしたね。マシン語プログラムにおけるレジスタの役割もこのようなものです。私たちは、レジスタとその使い方について知っておく必要があります。

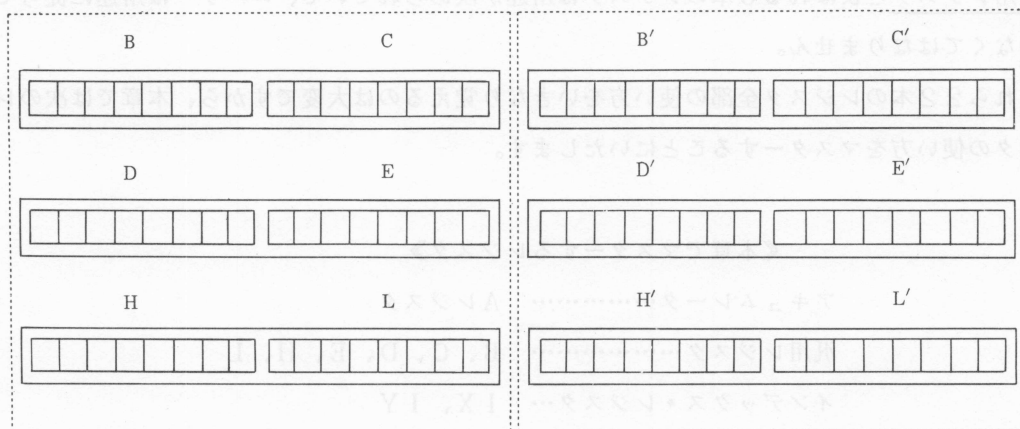
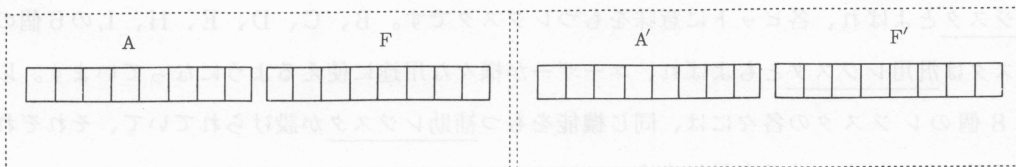
Ｚ８０ＣＰＵの内部には、種々の作業に使うための一時的な記憶回路がたくさんあります。このうち、１ビット単独で働くものはフリップフロップとよばれ、複数ビットを一組にして用いるものをレジスタとよびます。

フリップフロップの例としては、たとえばＩＦＦ(Ｉｎｔｅｒｒｕｐｔ Ｆｌｉｐ－Ｆｌｏｐ＝割り込みフリップフロップ)とよばれるものがありますが、本書の程度を越えますので、Ｚ８０ＣＰＵの解説書で学んで下さい。

さてレジスタに関してですが、ユーザーに公開されプログラム中で用いることのできるレジスタは全部で２２本あります。

次に掲げるレジスタ一覧表で、マス目の数はビット数を表わしています。

１６ビットレジスタ…ＩＸ，ＩＹ，ＳＰ，ＰＣ
８ビットレジスタ…Ａ，Ｆ，Ｂ，Ｃ，Ｄ，Ｅ，Ｈ，Ｌ
Ａ′，Ｆ′，Ｂ′，Ｃ′，Ｄ′，Ｅ′，Ｈ′，Ｌ′，Ｉ
７ビットレジスタ…Ｒ

各レジスタには、それぞれ個性があって、よいプログラムを作るには上手に使い分ける必要がありますが、本書を通じてマシン語プログラムを数多く作りながら、だんだんと身につけて

主レジスタ
補助レジスタ
A
F
A′
F′
B
C
B′
C′
D
E
D′
E′
H
L
H′
L′
I
R
インタラプト・レジスタ
リフレッシュ・レジスタ
IX
インデックス・レジスタX
IY
専用レジスタ
インデックス・レジスタY
SP
スタック・ポインタ
PC
プログラム・カウンタ

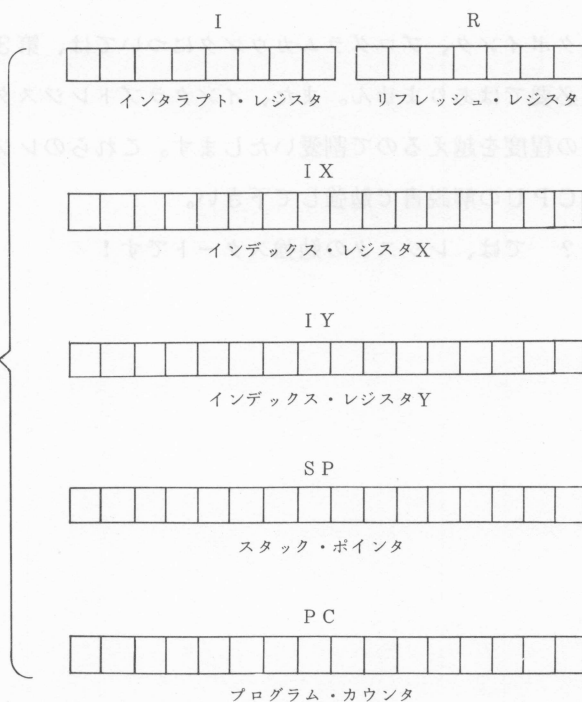



図2－1

いって下さい。

Aレジスタはアキュムレータともよばれることはすでに学びましたね。Fレジスタは、フラグレジスタとよばれ、各ビットに意味をもつレジスタです。B、C、D、E、H、Lの６個のレジスタは汎用レジスタともよばれ、ユーザーが様々な用途に使えるようになっています。以上の８個のレジスタの各々には、同じ機能をもつ補助レジスタが設けられていて、それぞれ′（ダッシュ）をつけて表示します。

専用レジスタとよばれる６本のレジスタは用途が決められていて、ユーザーは用途に従って使わなくてはなりません。

これら２２本のレジスタ全部の使い方をいきなり覚えるのは大変ですから、本章では次のレジスタの使い方をマスターすることにいたします。

《本章でマスターするレジスタ》

アキュムレータ…………… Aレジスタ
汎用レジスタ……………… B、C、D、E、H、L
インデックス•レジスタ… IX、IY

Fレジスタ、スタックポインタ、プログラムカウンタについては、第3章でマスターします。補助レジスタは当面必要ではありません。また、インタラプトレジスタ、リフレッシュレジスタに関しては、本書の程度を越えるので割愛いたします。これらのレジスタの使い方については、読者自らZ80CPUの解説書で勉強して下さい。

以上、よろしいですか？　では、レジスタの勉強スタートです！

2−2／データをレジスタに格納する

第１章においてと同様に、私たちは次の課題に取り組むことから始めましょう。

> 課題２　各レジスタに次の８ビット数値を格納するプログラムを作ること。
>
> | | |
> |---|---|
> | Ａレジスタ | ＡＡＨ |
> | Ｂレジスタ | ＢＢＨ |
> | Ｃレジスタ | ＣＣＨ |
> | Ｄレジスタ | ＤＤＨ |
> | Ｅレジスタ | ＥＥＨ |
> | Ｈレジスタ | １２Ｈ |
> | Ｌレジスタ | ３４Ｈ |
>
> ただし、プログラムはメモリー上　Ｄ０００Ｈ番地から格納するものとする。

まず、ニーモニック表記で考えてみます。前章において、私たちはＡレジスタに１バイト（＝８ビット）の数値を格納するロード命令　ＬＤ　というのを学びました。考え方はそれと全く同様です。次のようになりますね。

```
ニーモニック

LD   A,0AAH
LD   B,0BBH
LD   C,0CCH
LD   D,0DDH
LD   E,0EEH
LD   H,12H
LD   L,34H
RET
```

よろしいですか？　私たちはプログラムをモニターのＧコマンドで走らせることを前提にしていますから、最後に「プログラムを止めるおまじない」の　ＲＥＴ　を忘れないで下さいね。

次にこれらをハンドアセンブルで、マシンコードに変換してみましょう。前章でＡレジスタに関して行なったのと同様にして、付録の「Ｚ８０命令表」を用います。今度は、デスティネーション（データの送り先）がいろいろなレジスタに変わりますから注意して下さい。さて、各ニーモニックに対応するマシンコードは次のようになります。

《ニーモニック》	《マシンコード》
ＬＤ　Ａ，ｎ	３Ｅ　ｎ
ＬＤ　Ｂ，ｎ	０６　ｎ
ＬＤ　Ｃ，ｎ	０Ｅ　ｎ
ＬＤ　Ｄ，ｎ	１６　ｎ
ＬＤ　Ｅ，ｎ	１Ｅ　ｎ
ＬＤ　Ｈ，ｎ	２６　ｎ
ＬＤ　Ｌ，ｎ	２Ｅ　ｎ

従って、今の場合は、

```
マシンコード        ニーモニック

3EAA               LD   A,0AAH
06BB               LD   B,0BBH
0ECC               LD   C,0CCH
16DD               LD   D,0DDH
1EEE               LD   E,0EEH
2612               LD   H,12H
2E34               LD   L,34H
C9                 RET
```

図２－２

のようなマシンコードになりますね。最後にこれらをメモリーの　Ｄ０００Ｈ番地から配置してできあがりです。

課題２の解答

アト゛レス	マシンコート゛	ニーモニック
D000H	3EAA	LD A,0AAH
D002H	06BB	LD B,0BBH
D004H	0ECC	LD C,0CCH
D006H	16DD	LD D,0DDH
D008H	1EEE	LD E,0EEH
D00AH	2612	LD H,12H
D00CH	2E34	LD L,34H
D00EH	C9	RET

さっそくモニターを起動し、＊M　D０００　により、プログラムを　D０００H番地から格納しましょう。

```
MON
*M D000
:D000=3EAA
:D002=06BB
:D004=0ECC
:D006=16DD
:D008=1EEE
:D00A=2612
:D00C=2E34
:D00E=C9
:D00F=00
*
```

図２－３

では　＊G　D０００　により実行してみましょう。いかがですか？　＊印が表示され、カーソルが点滅し、モニターのコマンドレベルに戻ってプログラムは無事停止しましたね。結果を確かめましょう。はて？　そのためにはどのようにすればよいのでしょうか？

２－３／レジスタの中味を見る

私たちは前節で課題２を解決し、レジスタにデータを格納することに成功しました（そのはずですね）。そして、結果を確認するため「レジスタの中味を見る」問題に直面しています。

ところが……本節の標題に反するようですが、レジスタの中味を直接に見ることは実は不可能なのです。付録の「Ｚ８０命令表」をいくら探しても、そのようなマシン語はありません。困ったことですね。では結果の確認はできないのでは！？

御安心下さい。レジスタの中味は、直接には見ることはできませんが、間接的に見ることができるからです。第1章での課題1を思い出して下さい。そう！　メモリーに格納すればよいのですよ。

しかし、レジスタの内容をメモリーに格納するための命令は

LD　(nn′)，A

の形のものしかありませんでしたね。ソースはAレジスタと決まっていました。では、Bレジスタ以降の内容をメモリーに格納するにはどうしたらよいのでしょうか？

もう察しがついていると思いますが、答は意外に簡単です。そう、Aレジスタ経由でメモリーに転送すればよいのです。

かくして、私たちはレジスタ間のデータ転送命令を学ぶべき段階にまいりました。「Z80命令表」の8ビットロード命令の項で、デスティネーションがAレジスタである行を御覧ください。

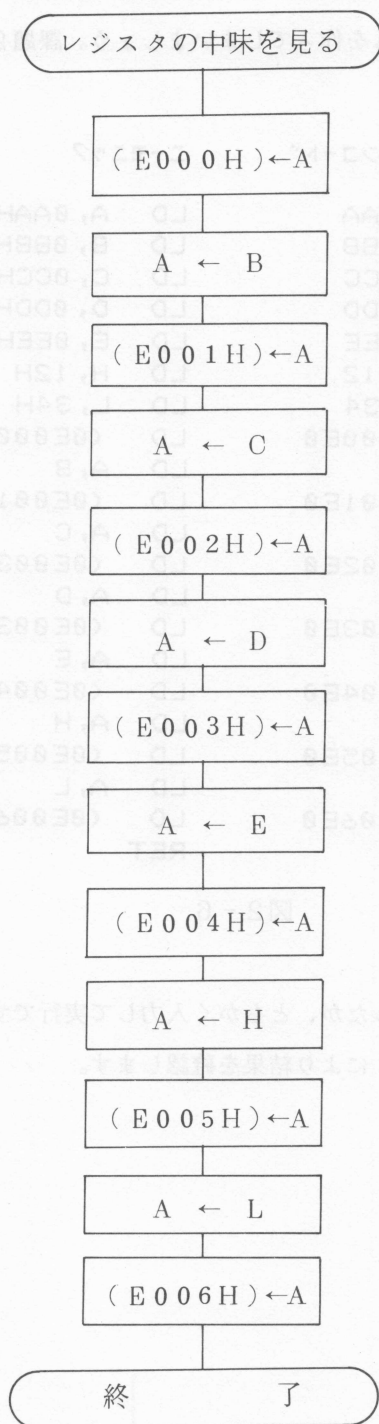
図2－4　《8ビットロード命令》

×	A	B	C	D	E	H	L	…………
LD　A，×	7F	78	79	7A	7B	7C	7D	…………

となっているはずです。ここに私たちの望む情報のすべてがあります。

データを格納するメモリーの番地は　E000H番地からにいたしましょう。処理の手順をフローチャートで示すと次のようになります。

図2－5

レジスタの中味を見る
（E000H）←A
A ← B
（E001H）←A
A ← C
（E002H）←A
A ← D
（E003H）←A
A ← E
（E004H）←A
A ← H
（E005H）←A
A ← L
（E006H）←A
終　　了

同じような処理の繰り返しが見られ、何か工夫する余地がありそうですが、それは後で考えることにして、さっそくプログラムを作ってしまいましょう。課題2のプログラムに続けて配置することにします。

```
ｱﾄﾞﾚｽ      ﾏｼﾝｺｰﾄﾞ      ﾆｰﾓﾆｯｸ

D000H      3EAA         LD   A,0AAH
D002H      06BB         LD   B,0BBH
D004H      0ECC         LD   C,0CCH
D006H      16DD         LD   D,0DDH
D008H      1EEE         LD   E,0EEH
D00AH      2612         LD   H,12H
D00CH      2E34         LD   L,34H
D00EH      3200E0       LD   (0E000H),A
D011H      78           LD   A,B
D012H      3201E0       LD   (0E001H),A
D015H      79           LD   A,C
D016H      3202E0       LD   (0E002H),A
D019H      7A           LD   A,D
D01AH      3203E0       LD   (0E003H),A
D01DH      7B           LD   A,E
D01EH      3204E0       LD   (0E004H),A
D021H      7C           LD   A,H
D022H      3205E0       LD   (0E005H),A
D025H      7D           LD   A,L
D026H      3206E0       LD   (0E006H),A
D029H      C9           RET
```

図2－6

少し長めのプログラムになりましたが、ともかく入力して実行です。できましたか？　実行が終わったら、　＊D　E000　により結果を確認します。

図2－7　《レジスタの中味を見る》

```
:E000=AA BB CC DD EE 12 34 00 /ｴｻﾌﾝ▀▄.4.
:E008=00 00 00 00 00 00 00 00 /........
:E010=00 00 00 00 00 00 00 00 /........
:E018=00 00 00 00 00 00 00 00 /........
:E020=00 00 00 00 00 00 00 00 /........
:E028=00 00 00 00 00 00 00 00 /........
:E030=00 00 00 00 00 00 00 00 /........
:E038=00 00 00 00 00 00 00 00 /........
:E040=00 00 00 00 00 00 00 00 /........
:E048=00 00 00 00 00 00 00 00 /........
:E050=00 00 00 00 00 00 00 00 /........
:E058=00 00 00 00 00 00 00 00 /........
:E060=00 00 00 00 00 00 00 00 /........
:E068=00 00 00 00 00 00 00 00 /........
:E070=00 00 00 00 00 00 00 00 /........
:E078=00 00 00 00 00 00 00 00 /........
```

入力ミスがなければ、結果は上図のようになっているはずです。実験成功ですね。こうして、私たちは間接的にではありますが、レジスタの中味を見ることに成功しました。

2－4／16ビットレジスタに挑戦

前節までに私たちは８ビットレジスタの基本的な使い方をマスターしたことになります。

次に、１６ビットレジスタであるインデックス・レジスタにアタックしましょう。またまた課題の登場です。

> 課題３　インデックスレジスタに次の１６ビット数値を格納するプログラムを作ること。
>
> ＩＸレジスタ ←── １２３４Ｈ
>
> ＩＹレジスタ ←── ５６７８Ｈ
>
> ただし、プログラムはメモリー上　Ｄ１００Ｈ番地から格納するものとする。

付録の「Ｚ８０命令表」で、１６ビットロード命令の項を御覧下さい。

図2－8《16ビットロード命令》

×	AF		IY	nn′	(nn′)
LD　IX, ×		…………		DD 21 n′ n	DD 2A n′ n
LD　IY, ×		…………		FD 21 n′ n	FD 2A n′ n

今の場合、デスティネーションはIXまたはIY、ソースは数値nn′ですから次のようになります。

《ニーモニック》　《マシンコード》

LD　IX, nn′　DD　21　n′ n

LD　IY, nn′　FD　21　n′ n

nn′は16ビット（＝2バイト）数値ですから、「上下位逆転の原則」を適用してマシンコードに変換される点を思い起こして下さい。

こうして、課題3の解答は次のようになります。

課題3の解答

```
アドレス    マシンコード    ニーモニック

D100H      DD213412       LD   IX,1234H
D104H      FD217856       LD   IY,5678H
D108H      C9             RET
```

8ビットレジスタの時と同様に、結果を確認するため、レジスタの内容をメモリーへ転送しましょう。今度は、E100H番地以降に転送することにいたします。

そのための命令を探して、再び１６ビットロード命令の項を見ると、次のようなマシン語が見つかります。

《ニーモニック》	《マシンコード》
ＬＤ（ｎｎ′），ＩＸ	ＤＤ　２２　ｎ′ｎ
ＬＤ（ｎｎ′），ＩＹ	ＦＤ　２２　ｎ′ｎ

やはり、アドレスを示す ｎｎ′ は「上下位逆転」でマシンコードに変換されます。

さあ、プログラムにまとめあげる番ですが、今回注意すべき点は、扱う数値が２バイトですから、ＩＸの内容を格納するのに２バイト分が必要で、Ｅ１００Ｈ番地と Ｅ１０１Ｈ番地を要することです。従って、ＩＹの内容は、Ｅ１０２Ｈ番地から２バイト分に格納されますね。

これらに注意して、ハンドアセンブルしたものが次のリストです。

```
アドレス    マシンコード    ニーモニック

D100H      DD213412      LD   IX,1234H
D104H      FD217856      LD   IY,5678H
D108H      DD2200E1      LD   (0E100H),IX
D10CH      FD2202E1      LD   (0E102H),IY
D110H      C9            RET
```

よろしいですか？ ではモニターを起動し、＊Ｍ Ｄ１００ によりプログラムを格納して下さい。終わりましたら、＊Ｇ Ｄ１００ で実行し、＊Ｄ Ｅ１００ で結果を確認して下さい。

図２－９ 《インデックス レジスタの中味を見る》

```
:E100=34 12 78 56 00 00 00 00 /4.xV....
:E108=00 00 00 00 00 00 00 00 /........
:E110=00 00 00 00 00 00 00 00 /........
:E118=00 00 00 00 00 00 00 00 /........
:E120=00 00 00 00 00 00 00 00 /........
:E128=00 00 00 00 00 00 00 00 /........
:E130=00 00 00 00 00 00 00 00 /........
:E138=00 00 00 00 00 00 00 00 /........
:E140=00 00 00 00 00 00 00 00 /........
:E148=00 00 00 00 00 00 00 00 /........
:E150=00 00 00 00 00 00 00 00 /........
:E158=00 00 00 00 00 00 00 00 /........
:E160=00 00 00 00 00 00 00 00 /........
:E168=00 00 00 00 00 00 00 00 /........
:E170=00 00 00 00 00 00 00 00 /........
:E178=00 00 00 00 00 00 00 00 /........
```

ダンプリストは図2－9のようになるはずです。何かお気づきになりませんか？　そう！数字が逆ですね。プログラムミスでしょうか？！

いいえ。正しい結果なのです。常識に反するようですが

Ｅ１００Ｈ番地　３４（下位）┐
　　　　　　　　　　　　　　├←── ＩＸ
Ｅ１０１Ｈ番地　１２（上位）┘
Ｅ１０２Ｈ番地　７８（下位）┐
　　　　　　　　　　　　　　├←── ＩＹ
Ｅ１０３Ｈ番地　５６（上位）┘

という形で格納されるのです。

これと似たようなことは前にも経験しています。そう、アセンブルにおける「上下位逆転の原則」でしたね。今回は、データ転送における上下位逆転の現象です。すなわち

《データ転送における上下位逆転の原則》

８０系ＣＰＵにおいては、２バイトデータを扱う命令で、上位１バイトと下位１バイトは逆転される。

とまとめられます。こういった「上下位逆転」は８０系ＣＰＵのマシン語では常につきまとってきます。慣れないうちは混乱したりしがちですが、図などに書き表わして理解するよう努めて下さい。

2－5／IXレジスタと間接アドレス指定

インデックスレジスタは、単に１６ビットレジスタというだけではなく、独特の機能を持っています。それは、メモリーのアドレスを指定する機能です。

たとえば、次の課題を考えてみて下さい。

課題４　隣接番地のメモリーにデータを格納すること。たとえば

Ｅ２００Ｈ番地 ←― データ　０ＡＨ

Ｅ２０１Ｈ番地 ←― データ　０ＢＨ

Ｅ２０２Ｈ番地 ←― データ　０ＣＨ

のように。プログラムは、Ｄ２００Ｈ番地から格納するものとする。

この課題の解答は、いろいろ考えられますが、ここではインデックスレジスタＩＸを用いてみましょう（ＩＹを用いても同様です）。必要とするマシン語は、「Ｚ８０命令表」の８ビットロード命令の項に出ています。

《ニーモニック》	《マシンコード》
ＬＤ　（ＩＸ＋ｄ），ｎ	ＤＤ　３６　ｄ　ｎ

ここで　ｎ　はもちろん数値を表わす１バイトデータですが、　ｄ　の方は初登場ですね。ｄ　は１バイト数値でディスプレイスメント（ｄｉｓｐｌａｃｅｍｅｎｔ＝変位の意）とよばれます。

今、ＩＸレジスタの内容が　Ｅ２００Ｈ　であるとします。すると、（ＩＸ＋ｄ）　は、Ｅ２００Ｈ番地からさらに　ｄ番地だけ離れた番地を示すことになります

番　地		
Ｅ２００Ｈ	…（ＩＸ＋００Ｈ）	ｄ＝００Ｈ
Ｅ２０１Ｈ	…（ＩＸ＋０１Ｈ）	ｄ＝０１Ｈ
Ｅ２０２Ｈ	…（ＩＸ＋０２Ｈ）	ｄ＝０２Ｈ
Ｅ２０３Ｈ	…（ＩＸ＋０３Ｈ）	ｄ＝０３Ｈ
Ｅ２０４Ｈ		

また、（ＩＸ＋ｄ）の括弧は　ＩＸ＋ｄ　の内容が示すメモリーの番地を表わします。先に出てきた　（ｎｎ′）　の（　）と同じです。ディスプレイスメントの働き、理解されました

か？　では課題４の解答を与えます。

《課題４のＩＸレジスタを用いた解答》

アドレス	マシンコード	ニーモニック
Ｄ２００	ＤＤ２１００Ｅ２	ＬＤ　ＩＸ，０Ｅ２００Ｈ
Ｄ２０４	ＤＤ３６０００Ａ	ＬＤ　（ＩＸ+００Ｈ），０ＡＨ
Ｄ２０８	ＤＤ３６０１０Ｂ	ＬＤ　（ＩＸ+０１Ｈ），０ＢＨ
Ｄ２０Ｃ	ＤＤ３６０２０Ｃ	ＬＤ　（ＩＸ+０２Ｈ），０ＣＨ
Ｄ２１０	Ｃ９	ＲＥＴ

これも　＊Ｍ　Ｄ２００　で入力し、＊Ｇ　Ｄ２００　で実行してみて下さい。結果は、＊Ｄ　Ｅ２００　Ｅ２０２　で確かめます。

《＊Ｄ　Ｅ２００　Ｅ２０２　で結果を確認》

```
:E200=0A 0B 0C 00 00 00 00 00 /........
```

このように、インデックスレジスタ等の１６ビットレジスタを用いて、アドレスを指定することを間接アドレス指定とよびます。これに対して、（ｎｎ’）のように１６ビット数値でアドレスを指定することを直接アドレス指定とよんでいます。

インデックスレジスタによる間接アドレス指定では、ディスプレイスメントを省略することができないこと［（ＩＸ）だけではいけません。このときも（ＩＸ＋００Ｈ）とする必要があります。］や、マシンコードがバイト数を多く要することなど欠点がありますが、利点もあります。それは、ディスプレイスメントを上手に利用することで、表形式のデータを扱うのに適していることや、また、ＩＸの内容を変えるだけで、データを格納する領域を容易に変更できる点などです。本書では以後、この方法は登場しませんが、有効な利用法を考えてみて下さい。

2－6／レジスタペア

汎用の８ビットレジスタである Ｂ，Ｃ，Ｄ，Ｅ，Ｈ，Ｌ は次のように組にして、１６ビットレジスタとして用いることができます。

ＢＣ，ＤＥ，ＨＬ

これらをレジスタペアとよんでいます。（たとえばＢＣレジスタペアというようによぶ。ペアを省いてＢＣレジスタとよぶこともある）

私たちは２－２節の課題２で

Ｂレジスタ ←――― ＢＢＨ

Ｃレジスタ ←――― ＣＣＨ

というようなことを行ないましたが、 Ｂ，Ｃ を８ビットレジスタとして単独に用いるのでなく、ＢＣレジスタペアとして

ＢＣレジスタペア ←――― ＢＢＣＣＨ

という１６ビットロード命令と考えることもできます。レジスタペアへの１６ビット数値のロード命令には次のものがあります。

《ニーモニック》	《マシンコード》
ＬＤ ＢＣ，ｎｎ′	０１ ｎ′ｎ
ＬＤ ＤＥ，ｎｎ′	１１ ｎ′ｎ
ＬＤ ＨＬ，ｎｎ′	２１ ｎ′ｎ

やはりアセンブル時に上下位逆転が生じていますから注意して下さい。

ＢＣレジスタペアを用いた次の実験を試みてみましょう。

アドレス	マシンコード	ニーモニック
D000	01CCBB	LD BC, 0BBCCH
D003	78	LD A, B
D004	3200E0	LD (0E000H),A
D007	79	LD A, C
D008	3201E0	LD (0E001H),A
D00B	C9	RET

＊M　D000　で入力、　＊G　D000　で実行、　＊D　E000　E001　で結果の確認をして下さい。

《＊D　E000　E001　による結果の確認》

```
:E000=BB CC 00 00 00 00 00 00 /ｻﾌ......
```

この実験により、LD　BC，nn′　は、　LD　B，n　と　LD　C，n′　を実行することと等価であることがわかりますね。他のレジスタペアについても全く同様です。ただ、　LD　BC，nn′　のマシンコードは3バイトで済むのに対し、　LD　B，n　と　LD　C，n′　の両方を行なうと4バイト必要ですから、レジスタペアを用いた方が省メモリーになります。

レジスタペアはこの他にも重要な機能をいくつか持っています。また　BC，DE，HL　の各々には個性があります。たとえば、第3章で登場する入出力装置とのデータのやりとりにはBCレジスタペアが活躍するとか、HLレジスタペアは16ビット演算で中心的な役割をはたすことなどがそうです。これらは後で学ぶことにして、本節ではもう1つの重要な機能である間接アドレス指定について考えましょう。

前節で私たちは、インデックスレジスタによる間接アドレス指定について学びましたが、レジスタペアを用いても同様のことができます。すなわち、

（BC）…BCレジスタペアで示される番地のメモリー
（DE）…DEレジスタペアで示される番地のメモリー
（HL）…HLレジスタペアで示される番地のメモリー

という形の間接アドレス指定ができます。

たとえば、HLレジスタペアを　E000H　、BCレジスタペアを　E001H　にセットしておくと、E000H番地の内容を　E001H番地へ転送するプログラムは以下のように書けます。

《レジスタペアを用いてメモリー間データ転送》

アドレス	マシンコード	ニーモニック
D000	2100E0	LD　　HL, 0E000H
D003	0101E0	LD　　BC, 0E001H
D006	7E	LD　　A, (HL)
D007	02	LD　　(BC), A
D008	C9	RET

＊M　D000　でプログラムを入力して下さい。次に、　＊M　E000　により初期データをセットしておきます（何でも構いません）。

```
*M E000
:E000=BB
:E001=CC
:E002=00
*
```

図2－9

これで実験準備はできました。　＊G　D000　で実行して下さい。

《＊D　E000　E001　で結果を確認》

```
:E000=BB BB 00 00 00 00 00 00 /ササ......
```

結果は上のようになるはずです。見事に、E000H番地の内容が、E001H番地へ転送されていますね。

メモリー間の転送は直接に行なうことはできません。必ずAレジスタ等のレジスタを用いて（つまりCPUを経由して）行なう必要があることを注意しておきます。

２－７／再び上下位逆転の注意

私たちは２－４節において、１６ビットレジスタの内容をメモリーに転送する命令

ＬＤ　（ｎｎ’），ＩＸ
ＬＤ　（ｎｎ’），ＩＹ

を学びました。本節では、１６ビットレジスタとして扱った時のレジスタペアについて同様のことを考えておきましょう。

「Ｚ８０命令表」の１６ビットロード命令を見ると次のマシン語が出ています。

《ニーモニック》	《マシンコード》
ＬＤ　（ｎｎ’），ＢＣ	ＥＤ　４３　ｎ’ｎ
ＬＤ　（ｎｎ’），ＤＥ	ＥＤ　５３　ｎ’ｎ
ＬＤ　（ｎｎ’），ＨＬ	２２　ｎ’ｎ

次のプログラムを考えます。

アドレス	マシンコード	ニーモニック
Ｄ０００	０１ＣＣＢＢ	ＬＤ　ＢＣ，０ＢＢＣＣＨ
Ｄ００３	１１ＥＥＤＤ	ＬＤ　ＤＥ，０ＤＤＥＥＨ
Ｄ００６	２１３４１２	ＬＤ　ＨＬ，１２３４Ｈ
Ｄ００９	ＥＤ４３００Ｅ０	ＬＤ　（０Ｅ０００Ｈ），ＢＣ
Ｄ００Ｄ	ＥＤ５３０２Ｅ０	ＬＤ　（０Ｅ００２Ｈ），ＤＥ
Ｄ０１１	２２０４Ｅ０	ＬＤ　（０Ｅ００４Ｈ），ＨＬ
Ｄ０１４	Ｃ９	ＲＥＴ

インデックスレジスタの時と同じで、２バイトずつデータを格納しますから、メモリーの番地は、　Ｅ０００Ｈ，　Ｅ００２Ｈ，　Ｅ００４Ｈ　と２バイトおきにする必要がありますね。

さて、これを入力し、実行する前に考えていただきたいのですが、Ｅ０００Ｈ～Ｅ００５Ｈ

番地の６バイト分のメモリー内容はどうなっていますか？　答えができたら、

＊Ｇ　Ｄ０００　で実行し、結果を確認して下さい。

《＊Ｄ　Ｅ０００　Ｅ００５　で結果を確認》

```
:E000=CC BB EE DD 34 12 00 00 /フサ▀ン4...
```

いかがですか？　予想通りの結果でしたか？　答えが合っていた方は、８０系ＣＰＵにおける「上下位逆転」について基本的な理解ができています。

Ｘ１のマシン語においても非常に大切なポイントですから、もう一度説明をしておきます。まず初心者の方が抱くであろう疑問としては（私も初めはそうでした！）、次のことがあろうかと想像いたします。

疑問　ＬＤ　（ｎｎ′），ＢＣ　は、（ｎｎ′）←ＢＣ　ということだから、 ２バイトデータ【ＢＣ】を、１バイト分のメモリー【（ｎｎ′）】に ロードしていて矛盾である？！

これはニーモニック上の約束ですから、仕方ないのですが、　ＬＤ　（ｎｎ′）、ＢＣ　の機能は　（ｎｎ′）←ＢＣ　ではありません。正しくは次のようになります。

解　説

ニーモニック：ＬＤ　（ｎｎ′），ＢＣ
【ｎｎ′は２バイト数値】
マシンコード：ＥＤ　４３　ｎ′ｎ
機　　　　能：（ｎｎ′）←Ｃ
（ｎｎ′＋１）←Ｂ

たとえば、プログラム中で用いた

ＬＤ　（０Ｅ０００Ｈ），ＢＣ

を実行するとデータの転送状況は次のようになります。

よろしいですか？　レジスタペアのレベルでも、上下位（Ｂレジスタが上位バイト、Ｃレジスタが下位バイト）の逆転が起こっていますね。　ＬＤ（ｎｎ′），ＤＥ　や　ＬＤ（ｎｎ′），ＨＬ　でも事情は全く同様です。

また、インデックスレジスタでも同じです。ＩＸレジスタの場合をまとめておくと、

解　説

ニーモニック：ＬＤ　（ｎｎ′），ＩＸ

マシンコード：ＤＤ　２２　ｎ′ｎ

機　　　能：（ｎｎ′）←ＩＸの下位バイト

（ｎｎ′＋１）←ＩＸの上位バイト

となっているのです。ＩＹレジスタでも同様です。

以上の注意点、よろしいですか？　先のプログラムの実行結果の予想が実際と合わなかった方は特に本節を綿密に読み直して理解しておいて下さい。

２−８／マシン語の構造を見る−１−

以上で、私たちはレジスタやメモリー間のロード命令の基本をマスターしたことになります。小さなものばかりでしたが、いくつかプログラム作りも経験してきました。このあたりで、マシン語の構造についてまとめておくのも有意義なことでしょう。

私たちが出会った各マシン語は、１バイト～４バイトのマシンコードで表わされていました。これはＺ８０マシン語のすべてについて言えます。すなわち、

Ｚ８０の全命令は、１バイトから４バイトまでのマシンコードで表わされる。

次に各マシン語の内部構造に注目しましょう。マシン語の最初の部分には、それがどのような処理操作をするものかを意味する部分が置かれます。これをオペレーション・コード（略してオペコード、ＯＰコードなどともいう）とよびます。この後には、命令の対象がどこなのか（どの番地のメモリーか、どの数値かなど）を示す部分が続きます。この部分をオペランドとよんでいます。各マシン語は［オペコードのみ］、あるいは、［オペコード＋オペランド］という構造をとっています。具体的に例で見ていきましょう。

１バイト命令は、［オペコードのみ］で意味を持ちます。例えば、

《マシンコード》	《ニーモニック》
７８	ＬＤ　Ａ，Ｂ

は１バイト命令で、７８　はオペコードです。

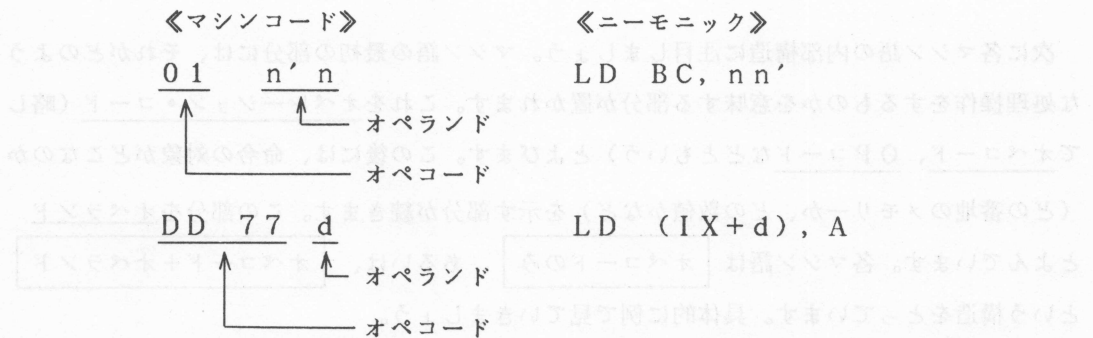
２バイト命令は、［オペコード＋オペランド］の形のものと、［２バイトで１つのオペコード］を意味するものがあります。例えば、

《マシンコード》	《ニーモニック》
３Ｅ　ｎ	ＬＤ　Ａ，ｎ
ED　４７	ＬＤ　Ｉ，Ａ（未出）

３Ｅ ← オペコード、ｎ ← オペランド

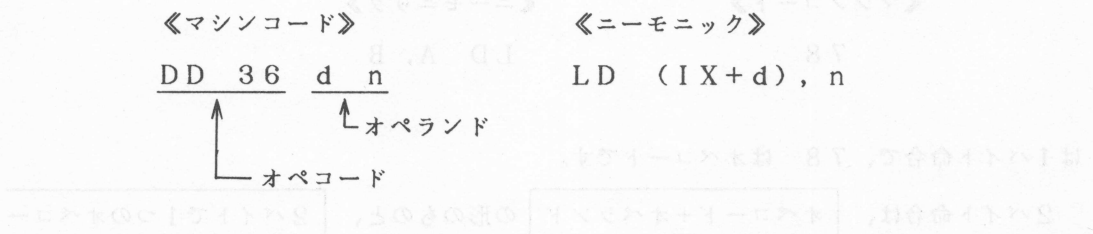
ED　４７ ← オペコード

などがそうです。

3バイト命令は、 オペコード＋2バイトオペランド の形のものと、 2バイトオペコード＋オペランド の形のものがあります。例えば、

4バイト命令は、 2バイトオペコード＋2バイトオペランド の形をとります。例えば、

《マシンコード》　　　　《ニーモニック》

DD 36 d n　　　　LD (IX＋d)，n

（DD 36：オペコード、d n：オペランド）

がその例ですね。

マシン語って多様なようでも、意外に構造が単純であることがわかると思います。むしろ、このように単純な命令だけで、コンピューターの多様な処理を実行できることを見抜いた天才 ――― マイクロプロセッサーの発明者 ―― 的な発想に驚嘆せざるを得ません！

2－9／マシン語の構造を見る―2―

さらにオペコードの内部構造を見てみましょう。私たちは、いくつかのロード命令に接して、マシンコードの決め方に何か秩序が潜んでいることを感じたはずです。たとえば次のよう

に並べてみると、はっきりしてきます。

《マシンコード》	《ニーモニック》
78	LD　A，B
79	LD　A，C
7A	LD　A，D
7B	LD　A，E
7C	LD　A，H
7D	LD　A，L

気まぐれにマシンコードを決めていたのでは、とてもこんな整然とはしないはずですね。では、なぜ　LD　A，B　はマシンコード　78　を持つのでしょうか。

解答は2進法のビットパターンの中に潜んでいます。16進数　78H　を2進法で表示してみましょう。

《16進数》	《2進数》
78	01111000

8ビットですから、0，1が8個並びますね。

さて、この8ビットを左（MSB）から2ビット、3ビット、3ビットと3つの部分に分けて下さい。

01	111	000

このようになりますね。まず最初の2ビット　01　は、　LD　の部分を意味しています。すなわち、CPUはマシンコード　78　を取り込んだ時、最初の　01　により、これが8ビットのレジスタ間転送命令であることを認識するのです。

では、次の3ビットずつは何を意味するのでしょう。実は、これはレジスタ番号を示しているのです。3ビットでは0～7の8通りの番号を表わすことができますね。8ビットのレジス

タには次のような番号がつけられています。

《レジスタ番号》　（２進法による）

０００＝Bレジスタ　１００＝Hレジスタ
００１＝Cレジスタ　１０１＝Lレジスタ
０１０＝Dレジスタ　１１０＝（HL）
０１１＝Eレジスタ　１１１＝Aレジスタ

レジスタ番号６（＝１１０）の（HL）というのは異常ですね。これは、レジスタではなく、HLレジスタペアで示される番地のメモリーですから。実は、このことは　Ｚ８０ＣＰＵの先祖である　８０８０ＣＰＵ　の名残りなのです。８０８０ＣＰＵ　では（HL）に相当するものを、メモリー上に仮想的に設定された“Mレジスタ”として扱っていたのです。Ｚ８０ＣＰＵ　は互換性を考慮して、　８０８０ＣＰＵ　の全命令をカバーした上で新機能を追加して設計されたために、このような名残りが見られるのです（２バイトオペコードを持つマシン語は　Ｚ８０　で追加された新機能です）。

これらのレジスタ番号を、　０１　の後にデスティネーション、ソースの順に並べるとマシンコードができます。いくつか例を示してみましょう

《ロード命令の構成》

０１　１１１　０００　→　７８Ｈ　は　ＬＤ　A,B
ＬＤ　　A　　B

０１　１１０　１１１　→　７７Ｈ　は　ＬＤ（HL），A
ＬＤ（HL）　A

０１　００１　１００　→　４ＣＨ　は　ＬＤ　C，H
ＬＤ　　C　　H

「Ｚ８０命令表」で確認すると合っていますね。では、　ＬＤ　（HL），（HL）　のような命令は作れるでしょうか？　同様に考えると、

０１　１１０　１１０　→　７６Ｈ

ＬＤ　（ＨＬ）（ＨＬ）

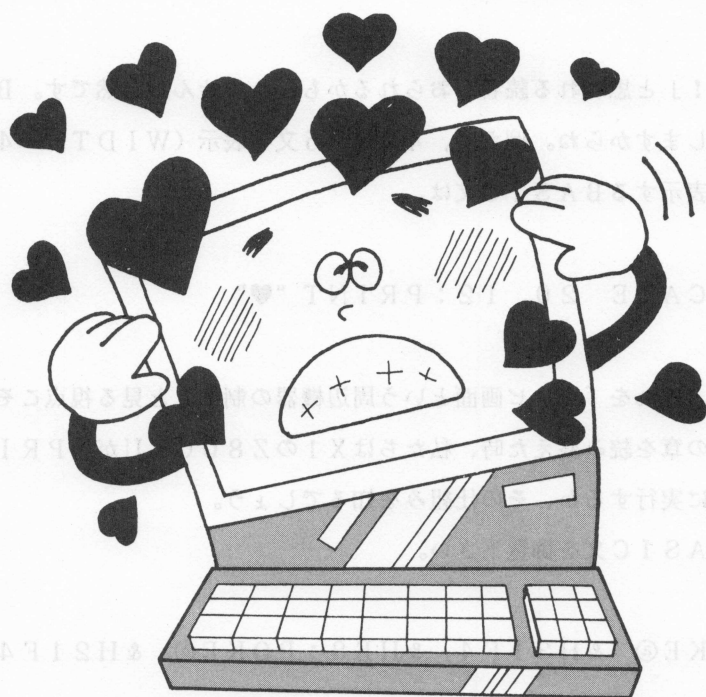
マシンコード　７６Ｈ　を持つ命令を探すと、次のように出ています。

解　説
ニーモニック：ＨＡＬＴ
マシンコード：　７６
機　　　能：ＣＰＵ停止命令。

恐ろしいですね！　このことを考えただけでも、メモリーからメモリーへの直接のデータ転送はＣＰＵを一時的にせよ停止させないとできないようになっているのですね。従って、メモリー間のデータ転送は、ＣＰＵ内のレジスタを経由して行なうようにマシン語が作られているのです。

いかがですか？　無味乾燥、天の声みたいに思えていたマシンコードが少しでも身近なものになれば、本節の目的は達せられたことになります。

次章からは、いよいよマシン語応用編のスタートです。頑張りましょう！

第3章　画面表示に挑戦！

■第３章　画面表示に挑戦！

３−１／♥表示にアタック！

前章までのウォーミング・アップを終えて、私たちはいよいよマシン語による最初の周辺機器制御　——　そうテレビ画面ですよ！　——　に挑戦しましょう！　当面の課題は、ナント！

テレビ画面への１文字表示

です。

「何を簡単な！」と思われる読者がおられるかもしれません。当然です。ＢＡＳＩＣでは基本中の基本に属しますからね。例えば、４０×２５文字表示（ＷＩＤＴＨ　４０）で、画面中央に♥マークを表示するＢＡＳＩＣ文は、

```
LOCATE 20, 12:PRINT "♥"
```

ですね。しかし、これを「テレビ画面という周辺機器の制御」と見る視点こそ、マシン語の視点なのです。この章を読み終えた時、私たちはＸ１のＺ８０ＣＰＵが、ＰＲＩＮＴ文にあたる作業をどのように実行するか、その仕組みを知るでしょう。

まず、次のＢＡＳＩＣ文を御覧下さい。

```
POKE@ &H31F4, &HE3:POKE@ &H21F4, 7
```

実は、この文は上のＰＲＩＮＴ文と全く同じ働きをします。「エッ？」と思われる方がいたら、画面をクリアした後、この文を実行してみて下さい。おわかりになりましたか？

ＰＯＫＥ@　というコマンドに初めて接した読者がおられても、それは当然です。なぜなら、これは「マシン語的」な色彩の強いＢＡＳＩＣコマンドだからなのです。

マニュアルの９１ページを見ますと、ＰＯＫＥ@　の機能は、

ＶＲＡＭ内の指定されたアドレスに１バイトのデータを書き込みます。 （　→　ＯＵＴ　）

と書いてあります。「ＶＲＡＭ」という言葉ご存知でしたか？

３－２／VRAMって何？

ＶＲＡＭとは、Ｖｉｄｅｏ　ＲＡＭ　の略称で、テレビ画面表示に用いられる特殊な読み書き自由メモリー（ＲＡＭ）のことです。メモリーである以上、きちんとアドレスがつけられていて、画面の各表示位置と１対１に対応するよう作られています。

シャープＸ１では、テキスト・モード（ＰＲＩＮＴ文などで文字を表示する）、グラフィック・モード（ＣＩＲＣＬＥ文などで図形を表示する）という２つの表示モードがあります。この各々に、ＶＲＡＭが付随しています。実は、グラフィック用のＶＲＡＭ（ＧＲＡＭなどともいう）は、本体には付属しておらず、オプションとして別途購入して、取り付けるようになっています。Ｘ１の持つ優れたグラフィック機能を活かすには、不可欠なデバイスなので、まだの方は是非取り付けていただきたいのですが、本章では、お持ちでない方のことも考慮し、テキスト・モードを中心に述べます。（Ｘ１Ｃ，Ｘ１ＤではＧＲＡＭは本体に付属しています。）

テキストＶＲＡＭは、　３０００Ｈ～３７ＦＦＨ　の範囲のアドレスを持っています。４０×２５モードでは、画面は横４０個、縦２５個のマス目に区切られ、各マス目がＶＲＡＭの各アドレスと対応します。図示すると次のようです。Ｘ１Ｃ，Ｄをお使いの方は、マニュアルに丁寧な記述があります(Ｘ１Ｃは１９２ページ～、Ｘ１Ｄは２１６ページ～)。

3000H	3001H	3002H	3003H	3004H	3005H	3006H				3025H	3026H	3027H
3028H	3029H	302AH	302BH	302CH	302DH	302EH				304DH	304EH	304FH
3050H	3051H	3052H	3053H	3054H	3055H	3056H				3075H	3076H	3077H
3078H	3079H	307AH	307BH	307CH	307DH	307EH				309DH	309EH	309FH
30A0H	30A1H	30A2H	30A3H	30A4H	30A5H	30A6H				30C5H	30C6H	30C7H
30C8H	30C9H	30CAH	30CBH	30CCH	30CDH	30CEH				30EDH	30EEH	30EFH
30F0H	30F1H	30F2H	30F3H	30F4H	30F5H	30F6H				3115H	3116H	3117H
3370H	3371H	3372H	3373H	3374H	3375H	3376H				3395H	3396H	3397H
3398H	3399H	339AH	339BH	339CH	339DH	339EH				33BDH	33BEH	33BFH
33C0H	33C1H	33C2H	33C3H	33C4H	33C5H	33C6H				33E5H	33E6H	33E7H

図３－１

［注］　マニュアル７２ページにもあるように、実は、４０×２５モードではテキスト画面は２ページ分あります。上図は、１ページ目のもので、全く同様にして、ＶＲＡＭアドレスの３４００Ｈ～３７Ｅ７Ｈ　が、２ページ目の画面と対応します。

ｘ、ｙ座標から、ＶＲＡＭアドレスを求めるのは、慣れないうちは大変です。そこで次のようなＢＡＳＩＣプログラムを作っておくと便利でしょう。

```
10 REM ── X,Y ｻﾞﾋｮｳ ｶﾗ VRAM ｱﾄﾞﾚｽ ｦ ﾓﾄﾒﾙ ──
20 '
30 INPUT "X,Y = ";X,Y
40 INPUT "ﾍﾟｰｼﾞ ( 1 or 2 ) = ";P
50 IF P<>1 AND P<>2 THEN 40
60 PRINT "VRAM ｱﾄﾞﾚｽ = ";HEX$(&H3000+(P-1)*&H400+X+Y*40)+"H"
70 PRINT :GOTO 30
```

実行してみましょう。

```
RUN
X,Y = ? 20,12
ﾍﾟｰｼﾞ ( 1 or 2 ) = ? 1
VRAM ｱﾄﾞﾚｽ = 31F4H

X,Y = ?
```

こうして、ｘ＝２０，ｙ＝１２のマス目に対応するＶＲＡＭアドレスは、３１Ｆ４Ｈであることがわかります。先の　ＰＯＫＥ＠　文で、

ＰＯＫＥ＠　<u>＆Ｈ３１Ｆ４</u>，＆ＨＥ３

＿＿＿部が、このアドレス（１６進数）です。

では、すぐ次にある　＆ＨＥ３　とは何でしょう。表示するには、少なくとも「どこに」、「何を」が必要ですが、ＶＲＡＭアドレスが「どこに」を示しますから、＆ＨＥ３　は「何を」に対応するものであると察しがつきますね。

３－３／表示の要素「何を」

コンピューターでは、文字・記号を規格化されたコード —— ＡＳＣＩＩコード（アスキー・コード） —— により表わします。ＡＳＣＩＩとは、

American Standard Code
for Information Interchange

の頭文字を並べたもので、英数字や記号に対して、１バイト（８ビット＝１６進２桁）のコードが決められています。日本には、さらにカナ文字がありますから、これもＪＩＳ（日本工業規格）により規格化されています。と言いたい所ですが、残念なことに、特にカナ文字に関しては、パソコンの機種により若干の違いがあるようです。他機種からのプログラムの移植に際しては注意が必要です。シャープＸ１では、マニュアル付属のＡＳＣＩＩコード表のように決められています。

話は戻りますが、第０章０－８節「メモリーの内容を見る」で、アスキーダンプという言葉が出てきたのを覚えていますか。たとえば、モニターを起動した後、＊Ｄ　０Ｆ４０　０ＦＤ８　とすると次のようなダンプリストが得られます。

```
:0F40=00 00 05 41 55 54 4F 0D /...AUTO.
:0F48=00 00 00 00 00 00 00 00 /........
:0F50=00 00 07 3F 54 49 4D 45 /...?TIME
:0F58=24 0D 00 00 00 00 00 00 /$.......
:0F60=00 00 03 4B 45 59 00 00 /...KEY..
:0F68=00 00 00 00 00 00 00 00 /........
:0F70=00 00 06 4C 49 53 54 1A /...LIST.
:0F78=0D 00 00 00 00 00 00 00 /........
:0F80=00 00 06 52 55 4E 20 20 /...RUN
:0F88=0D 00 00 00 00 00 00 00 /........
:0F90=00 00 06 4C 4F 41 44 20 /...LOAD
:0F98=0D 00 00 00 00 00 00 00 /........
:0FA0=00 00 06 57 49 44 54 48 /...WIDTH
:0FA8=20 00 00 00 00 00 00 00 / .......
:0FB0=00 00 05 43 48 52 24 28 /...CHR$(
:0FB8=00 00 00 00 00 00 00 00 /........
:0FC0=00 00 06 50 41 4C 45 54 /...PALET
:0FC8=20 00 00 00 00 00 00 00 / .......
:0FD0=00 00 05 43 4F 4E 54 0D /...CONT.
:0FD8=00 00 00 00 00 00 00 00 /........
```

図３－２

ここで右の方にある斜線／から右側がアスキーダンプの部分で、対応するメモリー内容をＡＳＣＩＩコードと見なした文字列を表示しています。これらの文字列はどこかで見た覚えがありますね。そうです、ファンクションキーの内容ですね。

このようにメモリー内容を１６進コードで眺めていては見落してしまうような文字列の情報をアスキーダンプの部分が教えてくれるのです。この部分は、ＢＡＳＩＣシステムプログラムの一部で、ファンクションキーのデータを格納しておく所です。従って、ファンクションキーの内容を変えようと思ったら、この部分に望むデータを入れればよいのです。

アスキーダンプでは、ＡＳＣＩＩコード　００Ｈ～１ＦＨ　の部分はピリオド・を用いて表示しています。これらは、コントロールコードと言って、文字には対応せず、改行したり、画面消去をしたり、ベルを鳴らしたりする動作に対応するコードです。アスキーダンプの説明は以上で終わりです。これでダンプリストの見方は完全にわかりましたね。

再び、私たちが現在直面している問題に眼を転じましょう。ＡＳＣＩＩコード表は次のように使います。

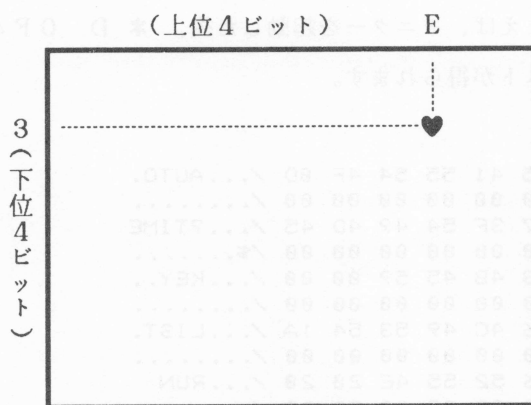



図３－３

ＡＳＣＩＩコード表で ♥マークを探し、図のようにして、♥のＡＳＣＩＩコードが、Ｅ３Ｈであるとわかります。こうして、懸案の

ＰＯＫＥ⓪　＆Ｈ３１Ｆ４，＆ＨＥ３

が、♥のコード　E3H　を、VRAMの　31F4H番地に書き込むことだとはっきりしました。

「では！」と、

POKE@　&H31F4，&HE3

とだけ実行してみて下さい。♥が出ましたか？

キチンと♥が出た方がおられたとしたら、それは運がよいのです。この時にはアドレスを&H3100～&H31FF　の範囲でいろいろ変えて実験してみて下さい。きっと♥以外の不可思議なマークが出るはずですから。

これは、一体どうしたことでしょう？！

3−4／アトリビュートとは？

本章の冒頭部に登場した　POKE@　文をもう一度思い出しましょう。

POKE@　&H31F4，&HE3：POKE@　&H21F4，7

マルチステートメントの後半＿＿＿部にカギがあるようですね。

結論を申し上げますと、実は画面表示の要素として、「何を」・「どこに」だけでは足りないのです。もう1つ必要です。そう「どのように」なのです！

文字・記号をどのように表示するか（何色か、反転させるか、点滅させるか等）に関わる情報を

アトリビュート（attribute＝属性）

とよびます。X1では、アトリビュートの指定は、1バイト（8ビット）で行ない、8個のビットは各々次の意味を持っています。

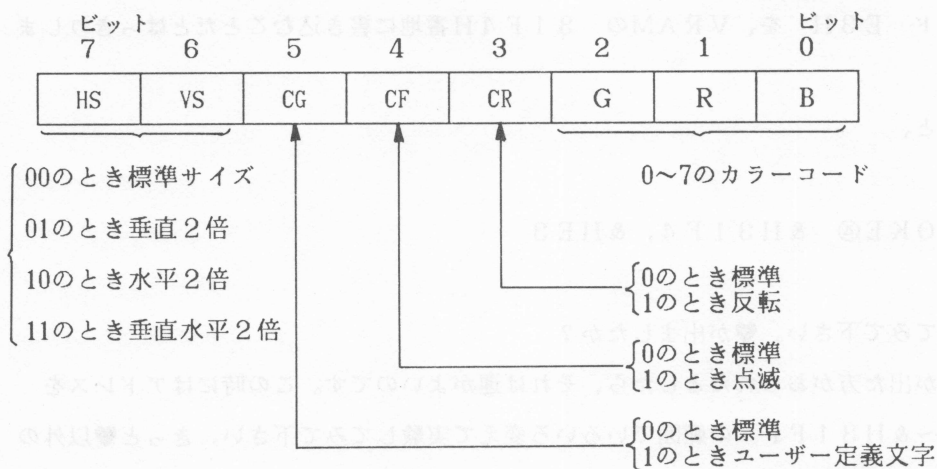



図３－４

ＶＲＡＭは画面表示用メモリーですから、アトリビュート指定にも当然関与します。ＶＲＡＭのうち、アトリビュート指定を受け持つ部分を、アトリビュートＶＲＡＭとよびます。Ｘ１では、アトリビュートＶＲＡＭは、　２０００Ｈ〜２７ＦＦＨ　というアドレスを持っていて、テキストＶＲＡＭに格納された文字データ各々にアトリビュート指定できるよう、次のように、テキストＶＲＡＭと１対１に対応しています。

テキストＶＲＡＭ	アトリビュートＶＲＡＭ
３０００Ｈ	２０００Ｈ
３００１Ｈ	２００１Ｈ
３００２Ｈ	２００２Ｈ
⋮	⋮
３７ＦＥＨ	２７ＦＥＨ
３７ＦＦＨ	２７ＦＦＨ

対応規則はおわかりですね。テキストＶＲＡＭアドレスの先頭の３を２に変えればよいのです。

先程、ＰＯＫＥⓐ　文の実験で、アドレスをいろいろ変えると♥以外の記号が現われた理由は、そのアドレスのアトリビュートが、垂直水平２倍文字などに指定されていたことによります。ですから、例えば、♥を標準モード、白色（ＣＯＬＯＲ　７）で表示するのなら、

00000111 = 10進数で7

標準モード　色コード7

とアトリビュート指定しなくてはなりません。これが、

POKE@ &H31F4, &HE3:POKE@ &H21F4, 7

後半＿＿＿部の意味です。

どうですか。ここで、次の公式を確認しておきましょう。

画面表示（テキスト・モード） ＝ テキストVRAMおよびアトリビュートVRAMの 対応するアドレスに各々のコードを書き込む。

コーヒー・ブレイク

実はこの公式、X1に初めて触れた（＝パソコン初体験）私にとって、最初の壁の1つでした。マニュアル168ページ（テキスト画面とその属性ポートへのアクセス方法）は、チンプンカンプンでした。皆様はいかがですか。この公式を理解することは、画面表示のマシン語による制御にとって偉大なる（？）一歩なのです。

3−5／OUTコマンドを用いて

前節で、私たちはテキスト画面の制御要素

どこに　　＝　テキストVRAMアドレス
何を　　　＝　ASCIIコード
どのように　＝　アトリビュートVRAMアドレスと
　　　　　　　アトリビュート指定コード

の理解に到達しました。ここまで来れば、マシン語による画面制御までもう一息です。
頑張りましょう！

私たちは、メモリーとＣＰＵがデータのやりとりをするためのマシン語 ―― ロード命令 ＬＤ ―― を学んできました。ＶＲＡＭもメモリーの一種ですから、ＬＤ命令で！ と思いたい所ですが、ここに最後の関門 ―― Ｘ１のハードウェアの壁 ―― が立ちはだかっています。

再び、ＰＯＫＥⓐ の話に戻りましょう。先にマニュアルの ＰＯＫＥⓐ の項を見た時、

（ →ＯＵＴ）

と記されていましたね。つまり、同様の結果はＢＡＳＩＣの ＯＵＴコマンド を用いて次のようにしても得られます。

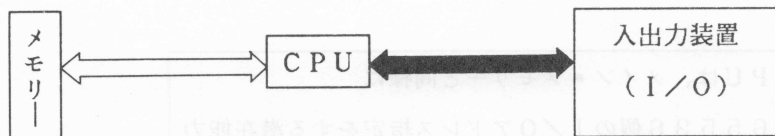
ＯＵＴ ＆Ｈ３１Ｆ４，＆ＨＥ３：ＯＵＴ ＆Ｈ２１Ｆ４，７

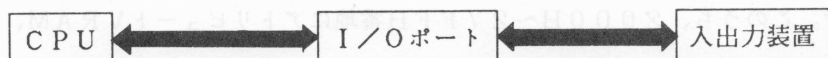
試してみて下さい。

マニュアル６２ページに ＯＵＴ コマンドについて次のように記されています。

機能	出力ポートに１バイトのデータを送ります。

ここに何げなく登場している「出力ポート」とは何のことでしょう？

3－6／I/Oポートの謎

第０章で確認した、コンピューターの基本構成をもう一度、ここで思い起こしましょう。本節のテーマは、図の右の⟷部分です。

ＣＰＵは、つながれている様々な入出力装置を制御しますが、それは通常、入出力ポート（I／Oポート）というものを間に介して行なわれます。ポート（ｐｏｒｔ）というと、私たちはすぐ「港」を連想しますが、辞書を引くと、「出入口」の意味もあり、コンピューター用語としての「ポート」は、こちらのようです。

I／Oポートには、メモリーと同様、１バイト単位の区画がアドレスをつけられて整然とならんでいます。I／OポートのアドレスをI／Oアドレスとよぶことがあります。

８ビットＣＰＵであるＺ８０では、（メイン）メモリーを最大６４Ｋバイトまで接続できるのでしたね。Ｚ８０は、これ以外に２５６個　（＝2^8）のI／Oアドレスを持つことができます　――　と通常のＺ８０の本には書いてあります。

シャープのＸ１では、クリーン設計で、メイン・メモリー６４Ｋバイトを自由に使用したいために、画面表示という特殊用途のメモリーであるＶＲＡＭをすべてI／Oポートに割りつけてあります。前節までに、しばしば登場した「ＶＲＡＭのアドレス」は、メイン・メモリー上のアドレスではなく、実はI／Oアドレスであったのです。

ここに重大な疑問があります。テキストＶＲＡＭのアドレスは、　３０００Ｈ～３７ＦＦＨでした。何と２０４８個ものアドレスがあります。

一方、Ｚ８０の普通の本によると、I／Oアドレスは２５６個だと書いてある。明らかにオ

カシイ！　のです。私も、Ｘ１のマシン語を勉強し始めた頃、この矛盾が最大の壁でした。

疑問は、Ｚ８０の少し詳しい本を読むことで解決しました。つまり、こういうことなのです。日ク、

> Ｚ８０ＣＰＵは、メイン・メモリーと同様に
> 2^{16}＝６５５３６個のＩ／Ｏアドレス指定をする潜在能力を持っている。普通は、下位８ビットのみを使用し、２５６個のＩ／Ｏアドレス指定をする。

と！　Ｘ１におけるＺ８０ＣＰＵの使用法は普通ではないのです！　シャープの設計陣が、どのようにして、ハードウェア的にＩ／Ｏアドレスの上位８ビットを活かしているのか、私にはわかりませんが、ともかく、このことがＸ１での入出力制御を理解する最大のキー・ポイントです！

以上をまとめると、Ｘ１でのＩ／Ｏアドレスは、００００Ｈ～ＦＦＦＦＨまである！　ということになります。このうち、２０００Ｈ～２７ＦＦＨ番地にアトリビュートＶＲＡＭ、３０００Ｈ～３７ＦＦＨ番地にテキストＶＲＡＭが割りつけられています。（ついでに述べておくと、４０００Ｈ～ＦＦＦＦＨ番地には、グラフィックＶＲＡＭが割りつけられています。）

３－７／入出力命令

前節で、私たちは、Ｘ１のハードウェア上の特色　──　Ｉ／Ｏアドレスが、メイン・メモリーのアドレスと同じ（６４Ｋバイト）だけある　──　を学びました。

Ｚ８０ＣＰＵが、あるアドレスを指定する時、それが、メイン・メモリーを選択しているのか、Ｉ／Ｏポートを選択しているのか明確にせねばなりません。メモリーとＣＰＵとの情報のやりとり（アクセス）は、ロード命令（ＬＤ）というマシン語で行なわれました。Ｚ８０では、Ｉ／Ｏポートのアクセスのために、別のマシン語　──　入出力命令（ＩＮ、ＯＵＴ）──　が用意されています。

本書付録の「Ｚ８０命令表」で入出力命令の項を御覧下さい。入力命令１２個、出力命令１２個の合計２４個ありますね。本節では代表として、

入力命令　　　　ＩＮ　Ａ，（Ｃ）

出力命令　　　　ＯＵＴ　（Ｃ），Ａ

を取り上げましょう。

まず、　ＯＵＴ　（Ｃ），Ａ　を考えましょう。ロード命令【ＬＤ】の時の感覚でいうと、この命令の機能は、

Ａレジスタの内容を、
Ｃレジスタの内容が指定するアドレスの
Ｉ／Ｏポートに出力する。

となりそうですね。［たとえば、　ＬＤ（ＨＬ），Ａ　を思い出して下さい！］

しかし、Ｃレジスタは８ビットのレジスタであり、２５６個までしかアドレスを区別できません。このことを強調するために、前節でくどい位に説明しておいたのでした。

私たちの予想に反して、Ｚ８０の少し詳しい本によると、正しくは、次のようになります。

解　説	
ニーモニック：	ＯＵＴ　（Ｃ），Ａ
マシンコード：	ＥＤ　７９
機　　能：	Ａレジスタの内容を、ＢＣレジスタペアの内容が指定するＩ／ＯアドレスのＩ／Ｏポートに出力する。

普通は、Ｂレジスタの指定する上位８ビットのアドレス情報を捨てる使用法をとるため、上記のように、Ｂレジスタを略したニーモニックが採用されているのでしょう。一般に流通する表記法ではありませんが、Ｘ１の特徴を考え、本書では、　ＯＵＴ　（Ｃ），Ａ　の機能を、

Ｉ／Ｏ（ＢＣ）　←　Ａ

で表わすことにします。［　ＬＤ　（ＨＬ），Ａ　の機能を　（ＨＬ）←　Ａ　と表記したの

の連想です。I／Oと付したのはI／Oアドレスを意味します。]

同様にして、　IN　A，(C)　は次のような機能を持ちます。

解　説	
ニーモニック：	IN　A，(C)
マシンコード：	ED　78
機　　　能：	A　←　I／O(BC)

これも　LD　A，(HL)　の機能を　A　←　(HL)　と表記したのの連想です。

3－8／♥表示をマシン語で！

大変お待たせいたしました。私たちは、たった♥1個を表示するために、これまで幾多の関門を越えてきました。VRAM，ASCIIコード，アトリビュート，I／Oポート，入出力命令　——　理解できましたか。かくして、私たちは、♥表示をマシン語で実行する準備をすべて整えました。

次のマシン語プログラムが、本章冒頭の課題の実現です。（メモリー上、D000H番地からプログラムを格納しました。）

図3－5

アドレス	マシンコード	ニーモニック	コ　メ　ン　ト
D000	3EE3	LD　A,0E3H	POKE@　&H31F4，&HE3
D002	01F431	LD　BC,31F4H	にあたる部分。
D005	ED79	OUT (C),A	
D007	3E07	LD　A,07H	POKE@　&H21F4，7にあた
D009	01F421	LD　BC,21F4H	る部分（アトリビュート出力）
D00C	ED79	OUT (C),A	
D00E	C9	RET	…プログラムの停止

モニターを起動し、Mコマンドで、D000H番地から、このプログラム（マシンコードの部分）を入力して下さい。終わったら、画面を消去した後、　＊G　D000　で、実行してみて下さい。

いかがですか？　画面中央に　♥　が出現し、＊で入力待ちとなりましたか？

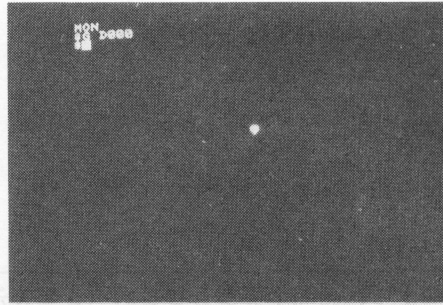



図３－６

☕　コーヒー・ブレイク

お疲れさまです。マシン語では、♥１文字の表示のためにも、いろいろなことを理解しておかねばならないことが、わかりましたか？　これをいとも簡単に行なうＢＡＳＩＣの　ＰＲＩＮＴ　コマンドって偉大だと思いませんか？

実を言うと、私にとっても、ここまでの道は苦難の道だったのです。ＰＣ－８００１のマシン語の本で、Ｚ８０マシン語を勉強していたのですが、ＰＣ－８００１では、ＶＲＡＭがメイン・メモリー上にあるので画面表示は、ＬＤ命令でよく、Ｘ１では、一体どうすればよいのか？　当時は、Ｘ１のマシン語の本もなく、想像を絶する苦闘の末、某ソフト会社のオール・マシン語ゲームの“ＧＡＭＥ　ＯＶＥＲ表示ルーチン”の解析により、理解に到達したのでした。それは、画面中央に　ＧＡＭＥ　ＯＶＥＲ　と表示するルーチンでした。そこに頻出する　ＥＤ　７９　というマシン・コードは何のためか？　の考察で秘密がわかったのです。私は、感動とともに、私のマシン語勉強ノートにこう書きました。

1983年　△月　○日
ゲーム「○○○○」の解読により
画面制御への道が開かれた。

さあ、皆さん！　次のステップへ向かって前進しましょう。

3－9／♥4個表示をめざして

余勢をかって、次の課題に取り組みます。

<u>課題</u>　40×25モードで、画面中央から♥を右へ4個表示せよ。
アトリビュートは白色標準モード。

テキストVRAMアドレスは、

ですから、前節のプログラムを4回繰り返せばよいですね。

図３－７

アドレス	マシンコード	ニーモニック	コ　メ　ン　ト
Ｄ０００	３ＥＥ３	LD A,0E3H	
Ｄ００２	０１Ｆ４３１	LD BC,31F4H	
Ｄ００５	ＥＤ７９	OUT (C),A	１つ目の♥を表示
Ｄ００７	３Ｅ０７	LD A,07H	
Ｄ００９	０１Ｆ４２１	LD BC,21F4H	
Ｄ００Ｃ	ＥＤ７９	OUT (C),A	
Ｄ００Ｅ	３ＥＥ３	LD A,0E3H	
Ｄ０１０	０１Ｆ５３１	LD BC,31F5H	
Ｄ０１３	ＥＤ７９	OUT (C),A	２つ目の♥を表示
Ｄ０１５	３Ｅ０７	LD A,07H	
Ｄ０１７	０１Ｆ５２１	LD BC,21F5H	
Ｄ０１Ａ	ＥＤ７９	OUT (C),A	
Ｄ０１Ｃ	３ＥＥ３	LD A,0E3H	
Ｄ０１Ｅ	０１Ｆ６３１	LD BC,31F6H	
Ｄ０２１	ＥＤ７９	OUT (C),A	３つ目の♥を表示
Ｄ０２３	３ＥＥ３	LD A,07H	
Ｄ０２５	０１Ｆ６２１	LD BC,21F6H	
Ｄ０２８	ＥＤ７９	OUT (C),A	
Ｄ０２Ａ	３Ｅ０７	LD A,0E3H	
Ｄ０２Ｃ	０１Ｆ７３１	LD BC,31F7H	
Ｄ０２Ｆ	ＥＤ７９	OUT (C),A	４つ目の♥を表示
Ｄ０３１	３Ｅ０７	LD A,07H	
Ｄ０３３	０１Ｆ７２１	LD BC,21F7H	
Ｄ０３６	ＥＤ７９	OUT (C),A	
Ｄ０３８	Ｃ９	RET	…プログラムの停止

上の解答例は、ＢＡＳＩＣでは、次のようなプログラムに相当します。

```
10 LOCATE 20,12 :PRINT "♥"
20 LOCATE 21,12 :PRINT "♥"
30 LOCATE 22,12 :PRINT "♥"
40 LOCATE 23,12 :PRINT "♥"
50 END
```

これでは、いかにも芸がないですね。また、♥の個数が１００個にでもなったらお手上げです。ＢＡＳＩＣでしたら、　ＦＯＲ　～　ＮＥＸＴ　によりループにすることをすぐ思いつきますね。

```
10 FOR X=20 TO 23
20    LOCATE X,12 :PRINT "♥"
30 NEXT X
40 END
```

では、マシン語でループ処理をするには、どうしたらよいでしょう。２つの問題点があります。

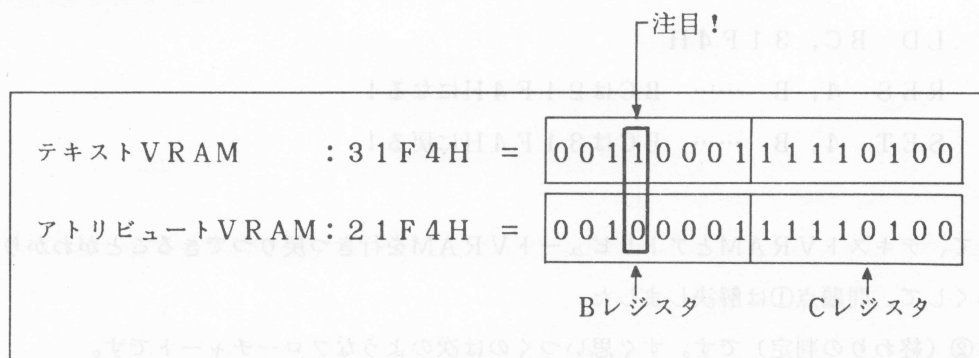
① ＶＲＡＭアドレス（ＢＣレジスタ）を１つずつ増やすには？
② ４回ループの後、終わりの判定はどうするか？

これらを解決するため、新しいマシン語を学びましょう。

解　説

ニーモニック：　ＩＮＣ　ＢＣ
マシンコード：　０３
機　　　能：　ＢＣレジスタペアの内容を１だけ増加させる。
［ＢＣ　←　ＢＣ＋１］

ＩＮＣは、　ｉｎｃｒｅｍｅｎｔ（増加）　からとったものです。　ＩＮＣ　ＢＣ　を用いることで、最初に１度だけ、ＢＣレジスタに値　３１Ｆ４Ｈ　を入れておけば、あとは１ずつ増やしてゆけばよいのです。これで①は解決！　と思うのは早計です。もう１つ、アトリビュートＶＲＡＭも、アドレス指定しなければなりませんでしたね。いろいろな方法があるでし

ょうが、次の賢明な方法を紹介します。テキストＶＲＡＭと、アトリビュートＶＲＡＭのアドレス対応を利用したものです。

例えば、画面中央にあたるＶＲＡＭアドレスの対応を御覧下さい。２つの違いは、Ｂレジスタに格納されるアドレス上位１バイトのうちの第４ビットが　１か０か　だけですね！

Ｂレジスタの第４ビット（右から５番目のビット）だけを、１にしたり０にしたりするマシン語はあるか？　と考えたくなります。実はあるのです。本書付録「Ｚ８０命令表」の「ビット操作命令」の項を御覧下さい。何と２４０個もの命令がありますね。しかし、驚くことはありません。これらは、どれか代表を理解すれば、あとは同様ですから。ここでは、次の２つをとり上げます。

ビット操作命令

解　説

ニーモニック：　ＲＥＳ　４，Ｂ

マシンコード：　ＣＢ　Ａ０

機　　　　能：　Ｂレジスタの第４ビットだけを０にする。

（ｒｅｓｅｔする）

解　説

ニーモニック：　ＳＥＴ　４，Ｂ

マシンコード：　ＣＢ　Ｅ０

機　　　　能：　Ｂレジスタの第４ビットだけを１にする。

（ｓｅｔする）

これらを用いると、

```
LD  BC，31F4H
RES  4，B  ……  BCは21F4Hになる！
SET  4，B  ……  BCは31F4Hに戻る！
```

となって、テキストVRAMとアトリビュートVRAMを行きつ戻りつできることがわかります。かくして、問題点①は解決しました。

次は②（終わりの判定）です。すぐ思いつくのは次のようなフローチャートです。

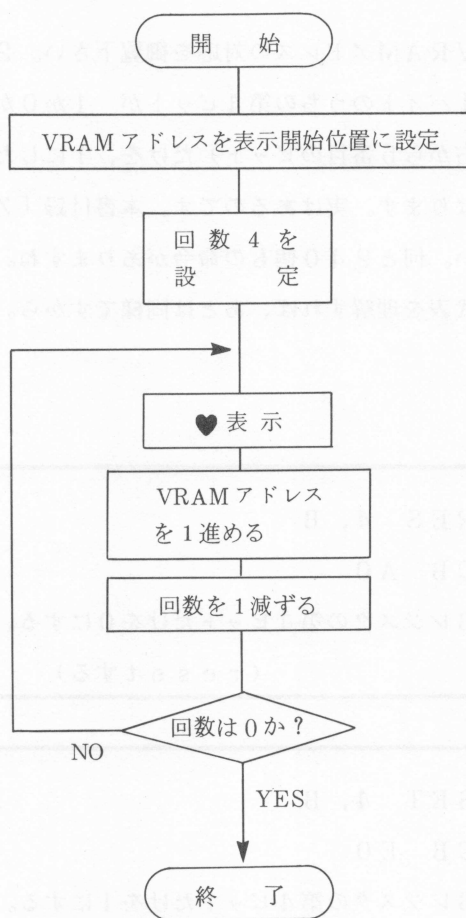



図3－8

条件判断が登場したことに注意して下さい。　BASIC　の　FOR　～　NEXT　ループでも、表には出ませんが、ループ終了の条件判断をしている訳ですね。では、マシン語で条件判断をするには、どうすればよいのでしょう。

3－10／マシン語での条件判断

話はひとまず、第2章に遡ります。そこでは、8ビット・16ビットの全レジスタが登場していました。このうち、前章で使ってみたのは、A，B，C，D，E，H，L，IX，IYの9個だけです。では、他のレジスタは何のためにあるのでしょう。これに関しては、おいおい考えていくことにして、本節では、フラグ・レジスタ　F　を取り上げます。

Fレジスタは、8ビットレジスタですが、他のレジスタと違って、LD命令で値を書き込むことはできません。では、何のためにあるかというと　──　そう、本節のテーマである条件判断のために設けられているのです。

フラグ（flag）というのは、「旗」のことですね。8本の旗が横一列に並んでいるのを想像して下さい。

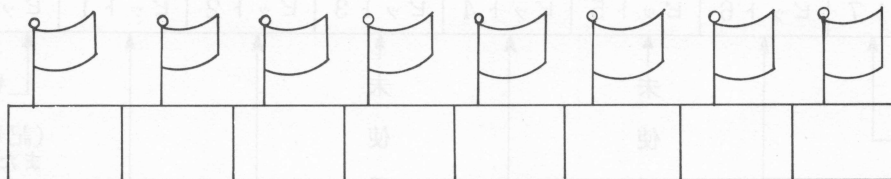



図3－9

これらの旗は、Fレジスタの8個のビットに対応します。ビットが1というのは、旗が立っている状態、ビットが0というのは、旗が降りている状態です。

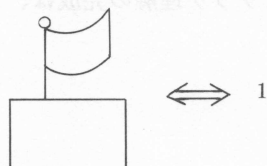
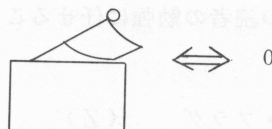



図3－10

これら８個のフラグは、ＣＰＵの動作に従って、パタパタとセット、リセットを繰り返すのです。コンピューター用語としてのフラグ、理解できましたか？

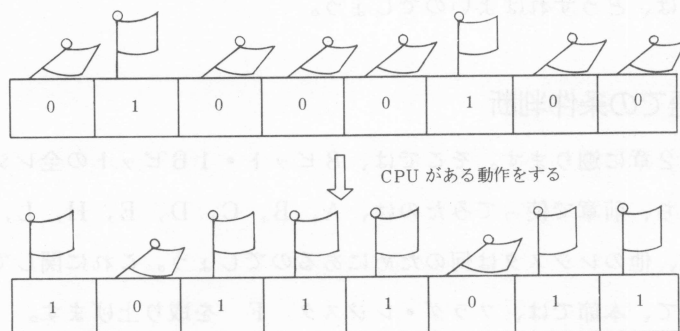
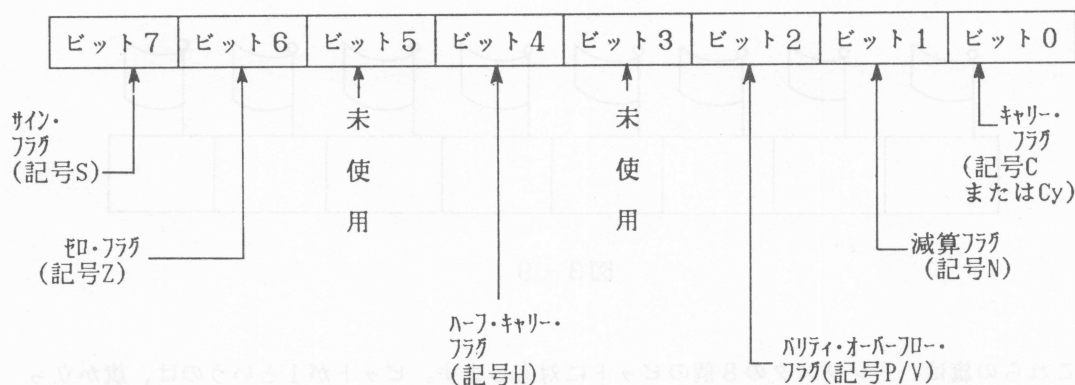



図３－１１

さて、８個のフラグ（ビット）には、各々特別な意味があり、その働きに応じて名前がつけられています。

使用される６個のフラグ全部を理解するのは、最初は大変です。フラグ理解の完成は、Ｚ８０ＣＰＵの本での読者の勉強に任せることとして、本書では、

ゼロ・フラグ　　（Ｚ）

キャリー・フラグ　（Ｃｙ）

の２つだけをマスターしましょう。

先程、Ｆレジスタは、条件判断のためにあると述べました。条件判断の中で、基本の１つは、２つの数の比較ですね（一方が他方より、大きいか、等しいか、小さいか）。マシン語で、２数の比較を行なうには、どうすればよいのでしょう。ここでは、２数は８ビットの数といたします（０～２５５）。

ＡとＢという２数を比較するのに、すぐ思いつくのは、　Ａ－Ｂ　を計算する方法です。ＣＰＵが、この符号（正か負か０か）を判断できれば、大小の比較ができる訳ですね。ここで、注意したいのは、「比較」という目的のためには、　Ａ－Ｂ　の答の数値自体は必要なく、その符号さえわかればよいということです。

実際、Ｚ８０ＣＰＵ　は、「比較命令」を持っていますが、概ね以上のような考え方で作られています。比較に必要なのは、　Ａ－Ｂ　の符号だと言いました。ＣＰＵは、フラグを変化させることにより、その情報を覚えておくのです。

Ａ，Ｂという８ビットの２数の比較でした。これを、「Ａレジスタの内容と、Ｂレジスタの内容を比較する」と読みかえると、比較命令の１つになります。

解　説

ニーモニック：　ＣＰ　Ｂ

マシンコード：　Ｂ８

機　　　　能：　Ａ－Ｂ　により、Ａレジスタの内容と、Ｂレジスタの内容の比較を行なう。ただし、Ａ－Ｂの答は出力せずフラグのみを変化させる。

- Ａ－Ｂ＝０　⇒　Ｚフラグ＝１
- Ａ－Ｂ≠０　⇒　Ｚフラグ＝０
- Ａ－Ｂ＜０　⇒　Ｃｙフラグ＝１
- Ａ－Ｂ≧０　⇒　Ｃｙフラグ＝０

ニーモニックの　ＣＰ　は、　ｃｏｍｐａｒｅ（比較する）　からとったものです。Ａレジスタの　Ａ　が省かれていますね。Ｚ８０では、比較される側は、必ずＡレジスタでなければならないと、決めてあるからです。これは、アキュムレータとしてのＡレジスタが持つ特権の１つです。

次に、フラグ変化ですが、ここでは、ゼロフラグとキャリーフラグに絞って見ることにします（他のフラグも変化しますが、ここでは関知しない立場をとります）。

たとえば、あらかじめ、Aレジスタに　１２Ｈ（比較されるもの）、Bレジスタに　３４Ｈ（比較するもの）が、セットされていたとします。ここで、ＣＰ　Ｂ　という命令を実行すると、

のようにフラグが変化します。では逆に、Aレジスタ＝３４Ｈ，Bレジスタ＝１２Ｈ　で、ＣＰ　Ｂ　を実行するとどうでしょう。答は、

となります。最後に、AもBも　１２Ｈ　であったらどうでしょう。フラグ変化は、

となります。

比較という操作も、暗黙のうちに減算という演算を行なっていますが、このようにＣＰＵが演算を実行したとき、２つのフラグは、

ゼロフラグ　　：　結果が0か否か、

キャリーフラグ：　加算での桁上げ（キャリー）、減算での借り（ボロー）
　　　　　　　　　が、生じたか否か、

という判断のために、主として用いられます。（A＜BのときA－Bを行なうと、8ビット目のもう1つ上の桁から借りが生じますね。ですから、Cyフラグがセットされたのです。）以上が、比較命令とフラグ変化の実際です。おわかりになりましたか？

行きがけの駄賃に、もう1つ比較命令を覚えておきましょう。　CP　B　と同様です。

解　説

ニーモニック：　CP　n

マシンコード：　FE　n　（nは8ビット数値）

機　　　　能：　Aレジスタの内容と、数値nとを比較する。
　　　　　　　　Aレジスタの内容は変化せず、フラグのみを
　　　　　　　　変化させる。

● A＝n　⇒　Zフラグ＝1
● A≠n　⇒　Zフラグ＝0
● A＜n　⇒　Cyフラグ＝1
● A≧n　⇒　Cyフラグ＝0

3－11／♥4個表示の完成

いよいよ、前々節（3－9）で掲げたフローチャートをマシン語で実現して、♥4個表示プログラムを完成させましょう。

まず、レジスタの役割を決めておきます。

Ａレジスタ	：	ＡＳＣＩＩコード、アトリビュートコードの出力、および比較のために用いる。
ＢＣレジスタペア	：	ＶＲＡＭのアドレス指定に用いる。

フローチャートを見ると、まだ何か足りませんね。そう、ループ回数のカウントです。これには、Ａ，Ｂ，Ｃ以外なら構いませんが、ここでは、Ｄレジスタをあてましょうか。

Ｄレジスタ	：	ループ回数のカウントに用いる。

そして、フローチャートにある「回数を１減ずる」を実現するために、次のマシン語を覚えましょう。

解　説

ニーモニック：　ＤＥＣ　Ｄ

マシンコード：　１５

機　　　　能：　Ｄレジスタの内容を１減ずる。フラグ変化は、

●Ｃｙフラグは変化しない。

●Ｚフラグは、結果が０なら１に、結果が０以外なら０に変わる。

［注］　Ｃｙフラグが不変というのは解せないかもしれませんが、「１だけ減ずる」という特別な減算命令だからだと、ここでは納得しておいて下さい。　ＤＥＣ　は、　ｄｅｃｒｅ－ｍｅｎｔ　の略。

回数のカウントができたら次はループさせるためのジャンプ命令がなくてはなりません。ＤＥＣ　Ｄ　によりＺフラグが変化することを利用し、次の条件ジャンプ命令を用います。

解 説

ニーモニック： JP NZ, nn′

［(注) nn′ はアドレスを指定する 16ビット数値（16進4桁）］

マシンコード： C2 n′ n

機　　　能： ゼロフラグが0ならば、nn′ 番地へ ジャンプする。そうでなければ何もしない。

ジャンプ先のアドレスを示す2バイト数値 nn′ をアセンブルする時は、「上下位逆転の原則」を適用するのでしたね。

以上で私たちは当面必要とするマシン語をすべて手に入れたことになります。さっそくプログラムを書き上げましょう。例によってプログラムはメモリー上 D000H番地から配置いたします。

図3－12 ループを用いた♥4個表示プログラム

アドレス	マシンコード	ニーモニック	コ メ ン ト
D000	01F431	LD BC,31F4H	…VRAM初期アドレス
D003	1604	LD D,04H	…ループ回数4
D005	3EE3	LD A,0E3H	♥を出力
D007	ED79	OUT (C),A	
D009	CBA0	RES 4,B	…アトリビュートVRAMへ移行
D00B	3E07	LD A,07H	アトリビュート出力
D00D	ED79	OUT (C),A	
D00F	CBE0	SET 4,B	…テキストVRAMへ復帰
D011	03	INC BC	…VRAMアドレスを1進める
D012	15	DEC D	…回数を1減ずる
D013	C205D0	JP NZ,0D005H	…条件ジャンプ
D016	C9	RET	…プログラムの停止

条件ジャンプの所、少し説明を加えておきます。 DEC D により、Dレジスタの内容は初期値4から1ずつ減ってゆきますが、0にならぬうちは、Zフラグは立ちませんから、

ＪＰ　ＮＺ，０Ｄ００５Ｈ　によりＣＰＵの制御は　Ｄ００５Ｈ番地へ移ります。こうしてループができるのですが、４回目に　ＤＥＣ　Ｄ　を実行すると、Ｄレジスタの内容が０になり、Ｚフラグが立つので、条件ジャンプはそのまま通過して、Ｄ０１６Ｈ番地の　ＲＥＴ　を実行し、モニターのコマンドレベルへ戻ってプログラムは停止します。

プログラムの理解ができましたら、モニターを起動し、マシンコードを入力して下さい。

できたら　＊Ｇ　Ｄ０００　で実行です。画面中央から右に　♥　が４個表示されましたか？

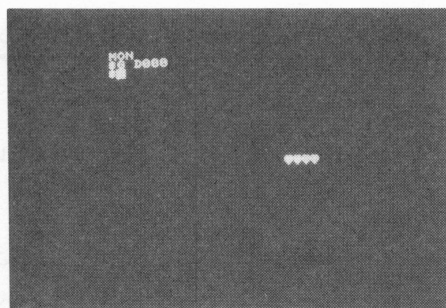



図３－１３

プログラム中、条件ジャンプ命令　ＪＰ　ＮＺ，ｎｎ′　が登場しましたが、よい機会ですから、ここでＺフラグとＣｙフラグによる条件の表記法をまとめておきます。

Ｚ　……「ゼロフラグが１であれば」

ＮＺ　……「ゼロフラグが０であれば」

Ｃ　……「キャリーフラグが１であれば」

ＮＣ　……「キャリーフラグが０であれば」

キャリーフラグの条件記号の　Ｃ　とレジスタの　Ｃ　を混同しないように注意して下さい。

付録の「Ｚ８０命令表」にある条件ジャンプ命令

ＪＰ　Ｚ，ｎｎ′

ＪＰ　Ｃ，ｎｎ′

ＪＰ　ＮＣ，ｎｎ′

の意味も、もうおわかりですね。

以上で、私たちは、マシン語による条件判断（フラグの使い方）を学び、ＢＡＳＩＣでの　ＦＯＲ　～　ＮＥＸＴ　にあたるループ処理をマスターしたことになります。ループ回数を変えたり（この方法で２５５回までループできます）、アトリビュートや表示位置を変えたりして、いろいろ実験してみて下さい。

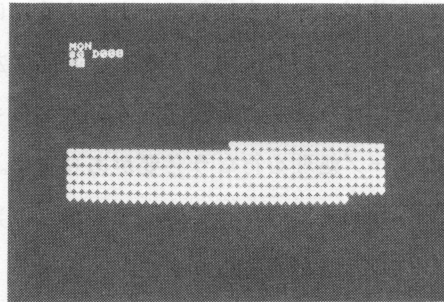



図３－１４

☕ティー・ブレイク

一息いれましょう。コーヒーばかりでは体によくなさそうなので、今回は紅茶にしました。ここまでの道のり、いかがでしたか？　フラグというコンピューター独特の概念を理解するまで、私も随分かかりました。プロの人たちが作ったプログラムを読むと、フラグを魔術のように用いて、省メモリー・高速化を実現しています。私には、まだそのような芸術的（ともいえる）プログラムは書けませんが、一歩一歩着実に前進してゆきましょう。

フラグと並んで、（私を含め）初心者にとって、難関となる概念に、スタックがあります。後で登場してきますから、その時は、心して取り組んで下さい。

３－12／画面反転プログラム

マシン語によるループ処理のまとめとして、１画面の色を反転するプログラムを作成しましょう。

まず、画面反転とは、どのようなものかを経験するために、テンキーの［－］キーを、［ＣＴＲＬ］を押しながら押して下さい。次に［ＣＬＲ］キーで、画面を消去してみて下さい。いかがですか？　画面が真白になりましたか？

これが１画面の反転です。もとに戻すには、再び［ＣＴＲＬ］＋ テンキー［－］を押して、［ＣＬＲ］して下さい。

Ｘ１では、どのようにして画面反転を実行しているのでしょう。

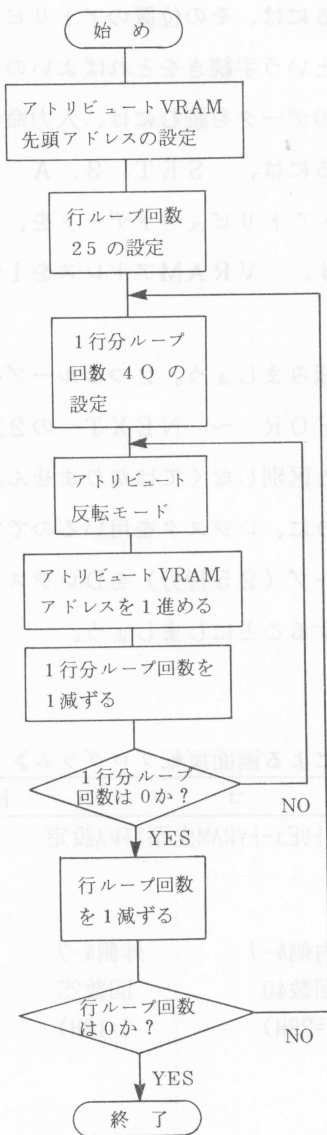
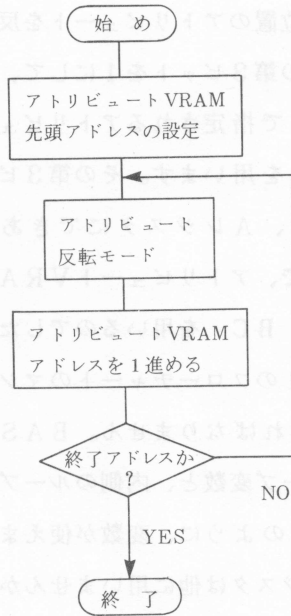
３－４節で、アトリビュートの勉強をしたとき、１バイトのアトリビュート情報の第３ビットが反転モードを意味すると述べました。

つまり、画面のある位置のアトリビュートを反転モードにするには、対応するアトリビュートＶＲＡＭの第３ビットを１にすればよいのです。これを、画面の左上隅から、右下隅まで全域にわたって繰り返せばよいことになります。１画面（４０×２５モード）は、１０００文字分ありますから、１０００回のループ処理を行なう必要があります。

前節でマスターしたのは、２５５回までのループ処理ですから、当然足りません。では、どうすればよいか？少なくとも２つの方法が考えられます。

方法１：　前節のループ処理を画面１行分（４０字）で使い、もう１つ行のループを考える２重ループ法。

方法２：　あらかじめ、終了アドレスを設定しておき、一回ごとに比較を行なう２バイトのアドレス比較法。

図3-15《フ ロ ー チ ャ ー ト》

これら２つの考え方は、いずれも大切ですから、２通りの方法で画面反転をしましょう。

いずれの方法でも、アトリビュートＶＲＡＭのアドレス指定は、ＢＣレジスタペアで行ないます。Ｘ１では、そのハードウェア上の特徴（３－６節参照）から、画面表示のために、　ＯＵＴ　（Ｃ），Ａ　命令を多用するので、ＢＣレジスタペアを頻繁に使います。

画面のある位置のアトリビュートを反転モードにするには、その位置のアトリビュートデータを読み、その第３ビットを１にして、再び出力するという手続きをとればよいのです。ＢＣレジスタペアで指定されるアトリビュートＶＲＡＭのデータを読むには、入力命令　ＩＮ　Ａ，（Ｃ）　を用います。その第３ビットを１にするには、　ＳＥＴ　３，Ａ　とすればよく、かくして、Ａレジスタにできあがった新しいアトリビュートデータを、　ＯＵＴ　（Ｃ），Ａ　で、アトリビュートＶＲＡＭへ出力します。　ＶＲＡＭアドレスを１つ進めるには、　ＩＮＣ　ＢＣ　を用いるのでしたね。

さて、方法１のフローチャートのマシン語化に取り組みましょう。２つのループの回数をカウントしなければなりません。ＢＡＳＩＣでも、　ＦＯＲ　～　ＮＥＸＴ　の２重ループでは、外側のループ変数と、内側のループ変数をきちんと区別しなくてはなりません。マシン語はＢＡＳＩＣのように、変数が使えませんが、かわりに、レジスタを用いるのです。今の場合、Ｄ，Ｅレジスタは他に用いませんから、外側のループ（２５行分）をＤレジスタで、内側のループ（１行４０字分）をＥレジスタで、カウントすることにしましょう。

図３－１６　《２重ループ（方法１）による画面反転プログラム》

アドレス	マシンコード	ニーモニック	コメント	
Ｄ０００	０１００２０	LD　BC,2000H	ｱﾄﾘﾋﾞｭｰﾄVRAM先頭ｱﾄﾞﾚｽ設定	
Ｄ００３	１６１９	LD　D,19H		
Ｄ００５	１Ｅ２８	LD　E,28H		
Ｄ００７	ＥＤ７８	IN　A,(C)	内側ﾙｰﾌﾟ	外側ﾙｰﾌﾟ
Ｄ００９	ＣＢＤＦ	SET　3,A	回数40	回数25
Ｄ００Ｂ	ＥＤ７９	OUT　(C),A	(=28H)	(=19H)
Ｄ００Ｄ	０３	INC　BC		
Ｄ００Ｅ	１Ｄ	DEC　E		
Ｄ００Ｆ	Ｃ２０７Ｄ０	JP　NZ,0D007H		
Ｄ０１２	１５	DEC　D		
Ｄ０１３	Ｃ２０５Ｄ０	JP　NZ,0D005H		
Ｄ０１６	Ｃ９	RET	ﾌﾟﾛｸﾞﾗﾑの停止	

上が完成したプログラムです。メモリーの　D000H番地から格納しました。2重ループの部分、よろしいですか？　モニターから入力して、実行して確認してみて下さい。

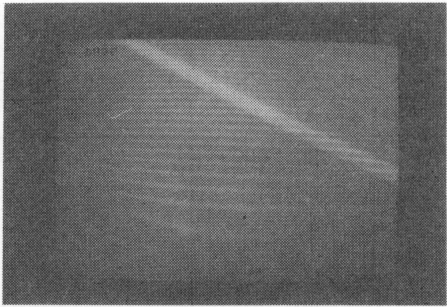
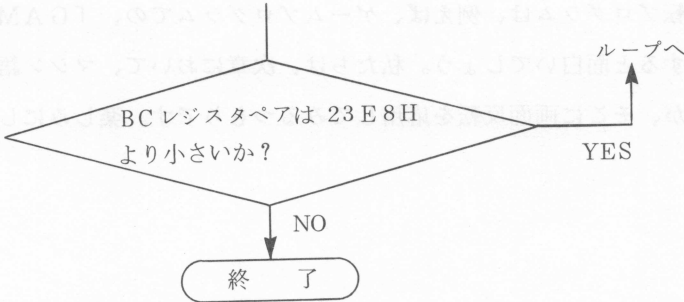



図3－17

では次に、方法2を考えます。この方法は、ループとしては一重、しかし、処理終了アドレスとBCレジスタペアを比較（2バイト比較）しなければなりません。（1ページ目の）画面右下隅に対応するアトリビュートVRAMのアドレスは、23E7H　です。このアドレスのアトリビュートを反転モードにした後、　INC　BC　により、アドレスは1つ進むはずですから、ループを抜け出す時には、BCレジスタペアは、23E8H　を示しているはずです。そこで、次のような条件判断を行なってみましょう。

付録の「Ｚ８０命令表」の比較命令の中には、私たちが必要とする２バイト（１６ビット）比較命令はありませんね。では、どうすればよいか？

答は意外に簡単です。１バイトずつ２段階に分けて比較すればよいのです。ただし、上位バイトがＢレジスタ、下位バイトがＣレジスタであることに注意して、Ｂレジスタと　２３Ｈの比較を先に行なわねばなりませんね。また、Ｂレジスタと数値を直接比較する命令もありませんから、まずＡレジスタにＢレジスタの内容を取り込まねばなりません。「小さければ」の条件判断はＣｙフラグで行なうのでした。

以上により、次のプログラムができました。

図３－１８　《アドレス比較（方法２）による画面反転プログラム》

アドレス	マシンコード	ニーモニック	コメント
Ｄ０００	０１００２０	LD　BC,2000H	
Ｄ００３	ＥＤ７８	IN　A,(C)	
Ｄ００５	ＣＢＤＦ	SET　3,A	
Ｄ００７	ＥＤ７９	OUT　(C),A	
Ｄ００９	０３	INC　BC	
Ｄ００Ａ	７８	LD　A,B	
Ｄ００Ｂ	ＦＥ２３	CP　23H	…Bを23Hと比較
Ｄ００Ｄ	ＤＡ０３Ｄ０	JP　C,0D003H	…小さければﾙｰﾌﾟへ
Ｄ０１０	７９	LD　A,C	
Ｄ０１１	ＦＥＥ８	CP　0E8H	…CをE8Hと比較
Ｄ０１３	ＤＡ０３Ｄ０	JP　C,0D003H	…小さければﾙｰﾌﾟへ
Ｄ０１６	Ｃ９	RET	…ﾌﾟﾛｸﾞﾗﾑの停止

例によって、入力、実行、確認をして下さい。予想どおりでしたか？

これらの画面反転プログラムは、例えば、ゲームプログラムでの、「ＧＡＭＥ　ＯＶＥＲ画面」などに応用すると面白いでしょう。私たちは、次章において、マシン語ゲームの作成に取り組む予定ですが、そこに画面反転を応用してみるつもりです。楽しみにしていて下さい。

[応用問題]

40×25 モードで、1画面消去のプログラムを作って下さい。

ヒント：文字を消すというのは、その位置に、空白（スペース）のASCIIコード 20H を出力することと同じです。

3−13／問題点の整理

前節において私たちは、画面反転のプログラムを完成いたしました。この種のプログラムは単独で用いるというよりは、ゲームプログラム等もっと大きなプログラムの中に組み込んで用いるという性質のものです。

私たちは、画面反転プログラムを仮にメモリーの D000H番地から配置しテストしましたが、上のことを前提に考えると「メモリーのどこに配置するか？」について、もう少し気を配る必要がありそうです。すなわち、画面反転プログラムを組み込むべき本体プログラムが決まらない以上、私たちはメモリーのどの番地に配置するか決定できないことになります。

これを裏返して言うと、どのような本体プログラムに組み込まれても、メモリーのどの番地に配置されても、正常な動作をすることが要請される訳ですね。このような視点で、私たちが作成した画面反転プログラムを反省してみましょう。方法1、方法2いずれの場合も、ループ処理のために条件ジャンプ命令（JP NZ，nn′ または JP C，nn′ ）を用いていますが、これらはジャンプ先のアドレスをきちんと指定してしまっているので、これらのプログラムは D000H番地から配置された時にのみ正常な動作をします。すなわち、私たちのプログラムでは上の要請に応えることはできない訳です。

メモリー上の配置番地を変更しても、正常な動作をするプログラムを、リロケータブル（relocatable＝再配置可能）であるといいます。この用語を用いると、前節での画面反転プログラムはリロケータブルではない！ といえます。すなわち、本節で提起される課題は次のものです。

課題 リロケータブルな画面反転プログラムを作ること。

３－14／リロケータブルとは？

本節では、アドレス比較（方法２）による画面反転プログラムを材料に、リロケータブルの問題を考えてゆきましょう。前節で指摘した通り、このプログラムをリロケータブルでなくしている原因は、２か所で使われている条件ジャンプ命令　ＪＰ　Ｃ，０Ｄ００３Ｈ　であることは明らかです。

もう少し具体的に説明してみましょう。条件ジャンプ命令　ＪＰ　Ｃ，０Ｄ００３Ｈ　が実行されると、条件成立時には常に　Ｄ００３Ｈ番地にジャンプしてしまいます。ではもし、この画面反転プログラムを　Ｅ０００Ｈ番地から配置して実行した時、ジャンプ先の　Ｄ００３Ｈ番地に「未知のマシン語」が書かれていたら、どうなるでしょう。そうです、何が起きるかわかりませんね。最悪の場合は「暴走」を起こすことでしょう！

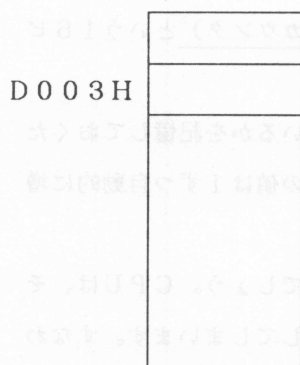
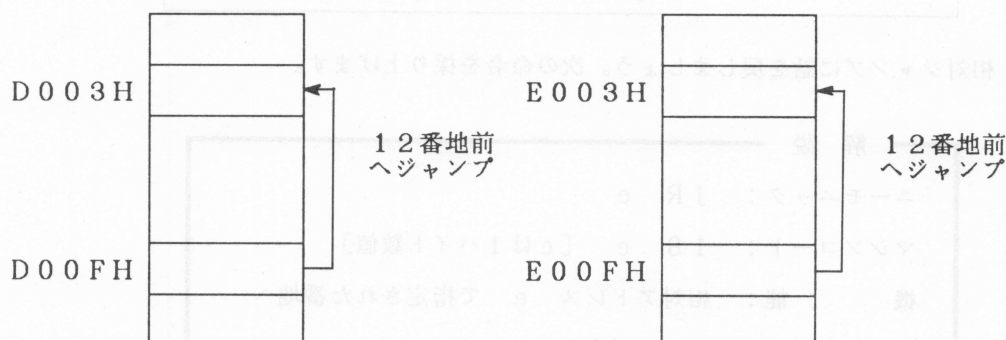
画面反転プログラムをたとえば　Ｅ０００Ｈ番地から配置しても、正常な動作をさせるためには、ジャンプ先のアドレスを　Ｄ００３Ｈ番地に固定してはならないのです。そのためにはどうすればよいのでしょうか？

私たちは、ある場所を指定するのに、どうしているでしょう。たとえば、住所なら、○○県○○市○○町○丁目○番地○号　などと指定しますね。郵便物は、こうして配達される訳です。

しかし、これだけでしょうか。旅行に出かけて、ある名所旧跡を訪れたいが、よくわからないとします。ガイドブックなどで、その近所までは何とか行けました。さて次は？

私たちは、土地の人たちや交番で尋ねます。「○○へ行きたいのですが、どうすればよいでしょう？」　すると、次のような答が大抵返ってくるはずです。「○○は、ここからその角を左へ曲がって、○○ｍ位行けばありますよ。」

マシン語においても、このような形式のアドレス指定があります。すなわち、あるアドレスを基準にして、（相対的に）目的のアドレスが、何番地後にあるのか、前にあるのか、という指定法です。これを、相対アドレス指定とよびます。これに対して、今まで使ってきた　○○番地　式のアドレス指定は絶対アドレス指定とよばれます。

２つのアドレス指定の違いを図にすると次のようになります。

私たちが、今、リロケータブルの問題と関係して必要としているのは、相対アドレス指定によるジャンプ命令 —— 相対ジャンプ命令という —— なのです。

上図により、相対ジャンプの形式でプログラムが書ければ、格納アドレスを変更しても、同じ結果が得られ、リロケータブルとなるのがわかります。

付録の「Z80命令表」のジャンプ命令の項を見ると、　JR　…　という形の命令がありますね。これらが相対ジャンプ命令です。（ニーモニックで、　JR　の部分は、　Jump Relative　の略です）

では、相対アドレスによりジャンプ先を指定するには、どうするのでしょう？　特に、どこを基準アドレスに採るのでしょう？

3－15／相対ジャンプ命令

第２章で、Ｚ８０の全レジスタを紹介した時、ＰＣ（プログラム・カウンタ）という１６ビットレジスタがあったことを思い出して下さい。

このレジスタは、ＣＰＵが今、メモリーのどのアドレスに注目しているかを記憶しておくために設けられています。通常、ＣＰＵが命令を実行するごとに、ＰＣの値は１ずつ自動的に増加されます。

では、ＰＣに、（１６ビットの）全く別の数値を入れたらどうなるでしょう。ＣＰＵは、その数値を次に実行すべきアドレスと認めて、そのアドレスに制御を移してしまいます。すなわち、ジャンプします。そうです！　ＪＰ　Ｃ，ｎｎ′　という命令は、Ｃｙフラグが立った時に、ＰＣに値　ｎｎ′　をロードする命令と考えることができる訳です。

ＪＰ　Ｃ，ｎｎ′　＝ ｛ Ｃｙフラグが立った時は、　ＰＣ←ｎｎ′
Ｃｙフラグが立たない時は、そのまま。

さて、相対ジャンプに話を戻しましょう。次の命令を採り上げます。

解　説

ニーモニック：　ＪＲ　ｅ

マシンコード：　１８　ｅ　［ｅは１バイト数値］

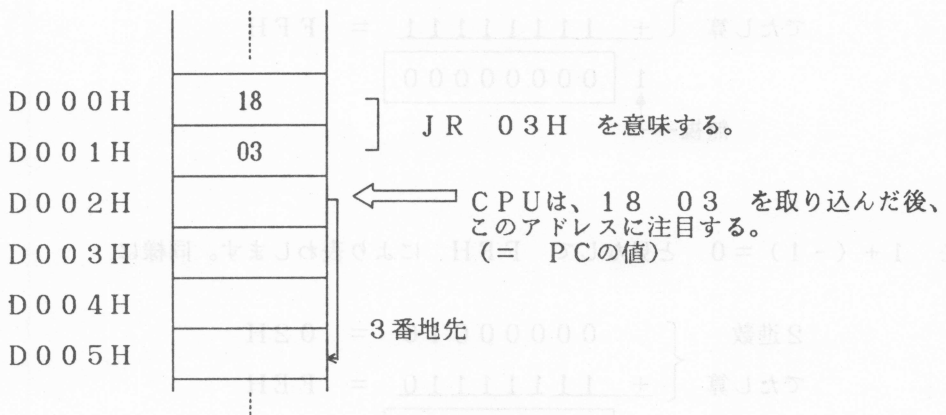
機　　　能：　相対アドレス　ｅ　で指定された番地へジャンプする。

相対アドレス　ｅ　の説明をいたします。

まず、基準にするアドレスですが、それは、

ＣＰＵが、ＪＲ　ｅ　という命令を取り込んだ直後のプログラム・カウンタの内容

です。もう少しかみくだきましょう。例えば、メモリーに次のように格納されていたとします。

《相対ジャンプのしくみ》

ＣＰＵは、命令を次々と実行して、Ｄ０００Ｈ，Ｄ００１Ｈ番地　に格納されている２バイト命令　ＪＲ　０３Ｈ　を取り込みました。すると自動的にプログラム・カウンタは１増えますから、この瞬間のＰＣは、　Ｄ００２Ｈ　を示しています。さて、　ＪＲ　０３Ｈ　はどういう命令かというと、ＰＣの値に、　０３Ｈ　を加えてしまうものなのです。すると、どうなりますか。ＰＣの値は、　Ｄ００２Ｈ＋０３Ｈ＝Ｄ００５Ｈ　となりますね。そこで、ＣＰＵは、３番地後の　Ｄ００５Ｈ番地へ制御を移します。つまり、　Ｄ００５Ｈ番地へジャンプします。おわかりですか？　一般に、

ＪＲ　ｅ　＝　ＰＣ　←　ＰＣ　＋　ｅ

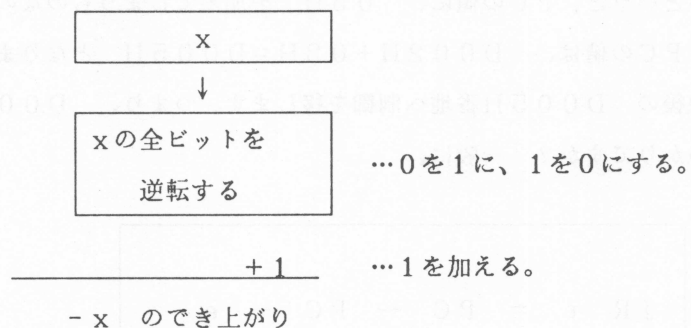
となって、　ＪＲ　ｅ　とはＰＣに関する加（減）算命令に他ならないのです！

では、手前にジャンプするには、どうすればよいのでしょう？　ｅ　が負の数であればよいですね。

８ビットで、負数を表わすには、２の補数という考え方をします。たとえば、　－１　を８ビットで表わすには、どうするかというと、

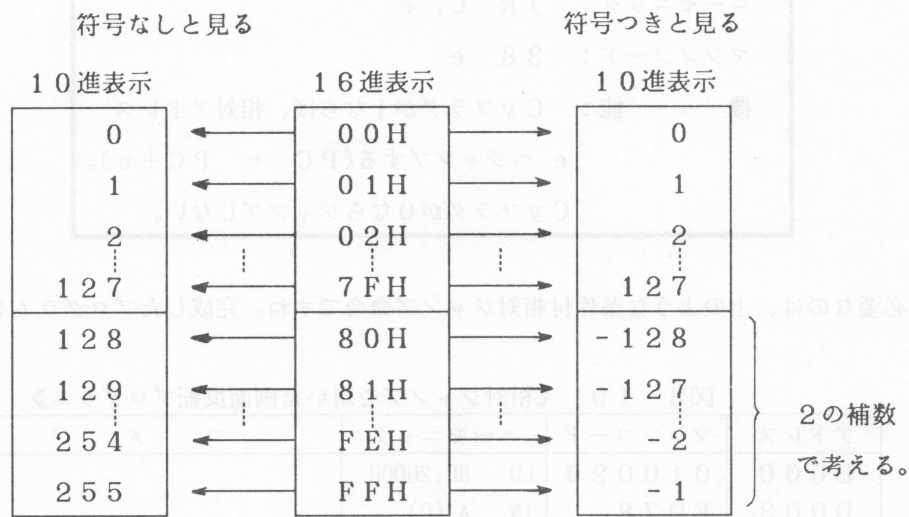
```
2進数      00000001 = 01H
でたし算  + 11111111 = FFH
        1 |00000000|
        ↑
       無視
```

を　１＋（－１）＝０　と見なして　ＦＦＨ　により表わします。同様に

ですから、－２　は　ＦＥＨ　で表わします。これらの作り方を反省すると、一般に、　ｘ　を８ビットの数とするとき、　－ｘ　を作るには、

となります。

さて、８ビットで正負を区別するには、その最上位ビット（ＭＳＢ）により行ないます。すなわち、符号付数と見なすには、

ＭＳＢ　＝　０　⇒　正（または０）の数
ＭＳＢ　＝　１　⇒　負の数

と決めています。こうして次の２通りの見方ができることになります。

話を戻します。相対ジャンプ　ＪＲ　ｅ　を考えていました。ここで、相対アドレスを表わす１バイト数値　ｅ　は、符号付の数と考えるのです（１０進数では　－１２８～＋１２７）。すなわち、３番地手前にジャンプするには、　－３　を２の補数で考えて、　ｅ＝ＦＤＨ　とすればよく、

ＪＲ　０ＦＤＨ　＝　３番地手前にジャンプ

となります。付録２に、「１バイト符号付１６進数」の表がありますから、確認し、よく練習しておいて下さい。

３－16／画面反転をリロケータブルに！

かくして、私たちは、画面反転プログラム（方法２で考えましょう）をリロケータブルにする方法をマスターしました。２つの条件ジャンプ命令を、相対ジャンプで置きかえてみましょう。

解　説

ニーモニック：　ＪＲ　Ｃ，ｅ
マシンコード：　３８　ｅ
機　　　　能：　Ｃｙフラグが１ならば、相対アドレス
ｅ へジャンプする(ＰＣ　←　ＰＣ＋ｅ)。
Ｃｙフラグが０ならジャンプしない。

必要なのは、上のような条件付相対ジャンプ命令ですね。完成したプログラムを掲げます。

図３－１９　《相対ジャンプを用いた画面反転プログラム》

アドレス	マシンコード	ニーモニック	コ　メ　ン　ト
Ｄ０００	０１００２０	LD　BC,2000H	
Ｄ００３	ＥＤ７８	IN　A,(C)	
Ｄ００５	ＣＢＤＦ	SET　3,A	
Ｄ００７	ＥＤ７９	OUT　(C),A	
Ｄ００９	０３	INC　BC	
Ｄ００Ａ	７８	LD　A,B	
Ｄ００Ｂ	ＦＥ２３	CP　23H	Bが23Hより小さければ12番地前
Ｄ００Ｄ	３８Ｆ４	JR　C,0F4H	… へジャンプ（PC=D00FH）
Ｄ００Ｆ	７９	LD　A,C	
Ｄ０１０	ＦＥＥ８	CP　0E8H	CがE8Hより小さければ17番地前
Ｄ０１２	３８ＥＦ	JR　C,0EFH	… へジャンプ（PC=D014H）
Ｄ０１４	Ｃ９	RET	… プログラムの停止

ニーモニックは、人間にとってわかりやすく記述するためにあります。上のプログラムのニーモニック表記では、相対アドレスをそのまま書いてありますが、これではどこにジャンプするのか見にくいですね。そこで、ニーモニック表記においては、相対ジャンプであっても相対アドレス値　ｅ　のかわりに、実際のジャンプ先を記すことがよく行なわれます（現実の多くのアセンブラでも受けつけてくれます）。以下に掲げるのはこの形で書いたリストです。

図３－２０

アドレス	マシンコード	ニーモニック	コ　メ　ン　ト
Ｄ０００	０１００２０	LD　BC,2000H	
Ｄ００３	ＥＤ７８	IN　A,(C)	
Ｄ００５	ＣＢＤＦ	SET　3,A	
Ｄ００７	ＥＤ７９	OUT　(C),A	
Ｄ００９	０３	INC　BC	
Ｄ００Ａ	７８	LD　A,B	
Ｄ００Ｂ	ＦＥ２３	CP　23H	
Ｄ００Ｄ	３８Ｆ４	JR　C,0D003H	…実際のジャンプ先
Ｄ００Ｆ	７９	LD　A,C	
Ｄ０１０	ＦＥＥ８	CP　0E8H	
Ｄ０１２	３８ＥＦ	JR　C,0D003H	…実際のジャンプ先
Ｄ０１４	Ｃ９	RET	…プログラムの停止

実際にリロケータブルであることを確認してみます。そのために、このプログラムをたとえば　Ｅ０００Ｈ番地へ転送してみましょう。

このような操作に便利なモニターコマンドがあります。

モニター　Ｔコマンド

《書式》　＊Ｔ　先頭アドレス　最終アドレス　転送先頭アドレス

《機能》　指定した範囲（先頭アドレスから最終アドレスまで）のデータを転送先頭アドレス以後に転送する。

ではモニターを起動し、次のように入力実行して下さい。

```
MON
*T D000 D014 E000
*█
```

図3－21

期待通り転送されているか確認してみます。予想だと、 E000H番地から E014H番地までに転送されているはずですから、 *D E000 E014 によりダンプします。

《*D E000 E014 による転送の確認》

```
:E000=01 00 20 ED 78 CB DF ED /.. ▞xヒ°▞
:E008=79 03 78 FE 23 38 F4 79 /y.x生#8ホy
:E010=FE E8 38 EF C9 00 00 00 /生X8□/...
```

いかがですか？ 予想通り転送されていますね。 E000H番地から格納された画面反転プログラムについて、ニーモニック表記をすると次のようなリストになります。

図3－22 《E000H番地から格納》

アドレス	マシンコード	ニーモニック	コ メ ン ト
E000	010020	LD BC,2000H	
E003	ED78	IN A,(C)	
E005	CBDF	SET 3,A	
E007	ED79	OUT (C),A	
E009	03	INC BC	
E00A	78	LD A,B	
E00B	FE23	CP 23H	
E00D	38F4	JR C,0E003H	…実際のジャンプ先
E00F	79	LD A,C	
E010	FEE8	CP 0E8H	
E012	38EF	JR C,0E003H	…実際のジャンプ先
E014	C9	RET	…プログラムの停止

相対アドレスから実際のジャンプ先を計算すると、Ｅ００３Ｈ番地になりますね。つまり、このプログラムは、Ｅ０００Ｈ番地から配置しても、Ｄ０００Ｈ番地から配置したのと同じ働きをすることになります。こうして、私たちは画面反転プログラムをリロケータブル化することに成功しました！

３−17／サブルーチンとスタック

画面反転プログラムの完成をめざしてきた私たちの試みは前節において、一応満足できる段階に達したことになります。そこで本節以降は、このプログラムに関して唯一不透明になっている部分 —— 第１章以来の宿題 —— である「プログラム停止のためのおまじない　Ｃ９　」に関して解明していくことにいたします。

まず、私たちがすでに慣れ親しんでいるモニターのＧコマンドについて、マニュアルの正式の説明を読むことにいたします。マニュアル１７６ページを御覧下さい。

＊Ｇ　ゴーサブ

＊Ｇ　コールアドレス

指定したコールアドレスをサブルーチンコールします。
スタックポインターは、ＦＦＦＥ（１６進）にあります。

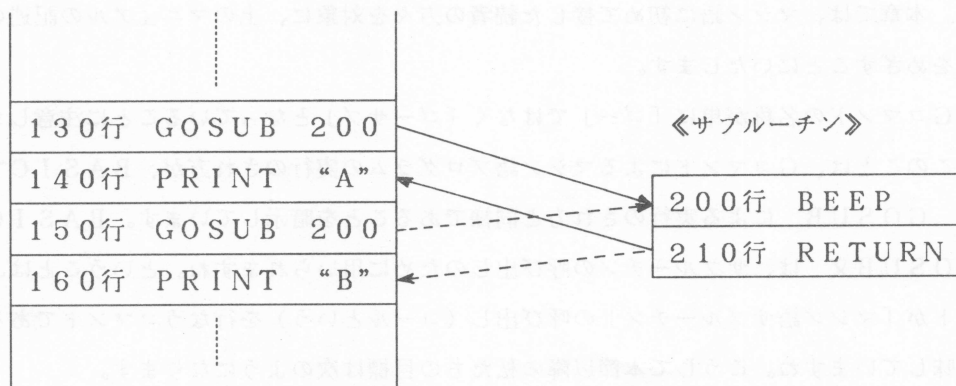
とありますが、おわかりですか？　この記述だけでわかる方は、マシン語について基礎知識のある方と思われますので、本節以降の本章の内容は読みとばして構いません。第４章へ進んで下さい。本章では、マシン語に初めて接した読者の方々を対象に、上のマニュアルの記述の完全理解をめざすことにいたします。

まずＧコマンドの名称が単に「ゴー」ではなく「ゴーサブ」となっていることに注意しましょう。このことは、Ｇコマンドによるマシン語プログラムの実行のされ方が、ＢＡＳＩＣで言う所の　ＧＯＳＵＢ　による実行のされ方と同様であることを暗示しています。ＢＡＳＩＣでは　ＧＯＳＵＢ文　は、サブルーチンの呼び出しのために用いられますね。ということは、Ｇコマンドが「マシン語サブルーチン」の呼び出し（コールという）を行なうコマンドであることを意味していますね。こうして本節以降の私たちの目標は次のようになります。

目標　マシン語サブルーチンとその実行の原理について 　　　理解すること。

ＢＡＳＩＣにおけるサブルーチンとその実行のされ方について復習しましょう。次に簡単なＢＡＳＩＣプログラムを掲げます。

```
100 REM ―― BASIC ノ サブルーチン ――
110 '
120 CLS
130 GOSUB 200
140 PRINT "A"
150 GOSUB 200
160 PRINT "B"
170 END
180 '
190 '
200 BEEP
210 RETURN
```

行番号１７０までがメインルーチン、行番号２００以降がサブルーチンですね。実行のされ方は次図のようになります。

ＢＡＳＩＣインタプリターは、ＧＯＳＵＢ文に出会うと戻り行をメモリー上のどこかに保存し、それから　ＧＯＳＵＢ文の示すサブルーチンへジャンプしてゆきます。さて、サブルーチンの末尾へ来て、　ＲＥＴＵＲＮ　に出会うと保存しておいた戻り行を復帰させ制御をそこに移す　――　以上が、　ＧＯＳＵＢ　によるサブルーチン実行の原理です。

ＢＡＳＩＣプログラムにおいては、ＢＡＳＩＣインタプリターがすべてを管理してくれますが、私たち自身が作成するマシン語プログラムでは、ＢＡＳＩＣインタプリターの手を借りずに直接ＣＰＵに働きかける点が異なっています。しかし、マシン語においてもサブルーチンへのジャンプとメインルーチンへの復帰の原理は全く同様です。

ＣＰＵが、次々とマシン語命令を実行していく過程で、サブルーチンからの戻り番地のように保存しておくべきデータがいくつか生じてきます。データの一時的保管場所であるレジスタでは数に限りがありますから、このような場合は、メモリーの一部を保管場所として割りあてるということをいたします。この目的に用いられるメモリー上の領域をスタックとよびます。スタック（ｓｔａｃｋ）を辞書で引くと、「干し草の山、積み重ね」と出ています。コンピューターの場合、「積み重ね」られるものは（保存しておきたい）データですが、次のようなイメージでとらえるとわかりやすいでしょう。

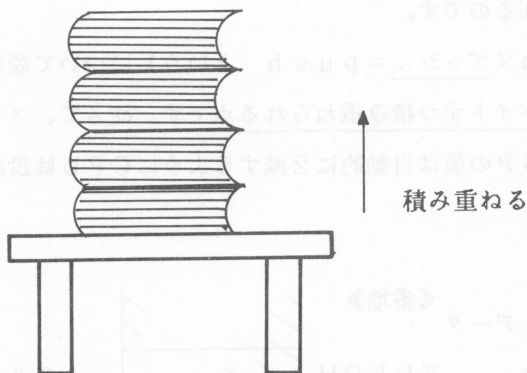
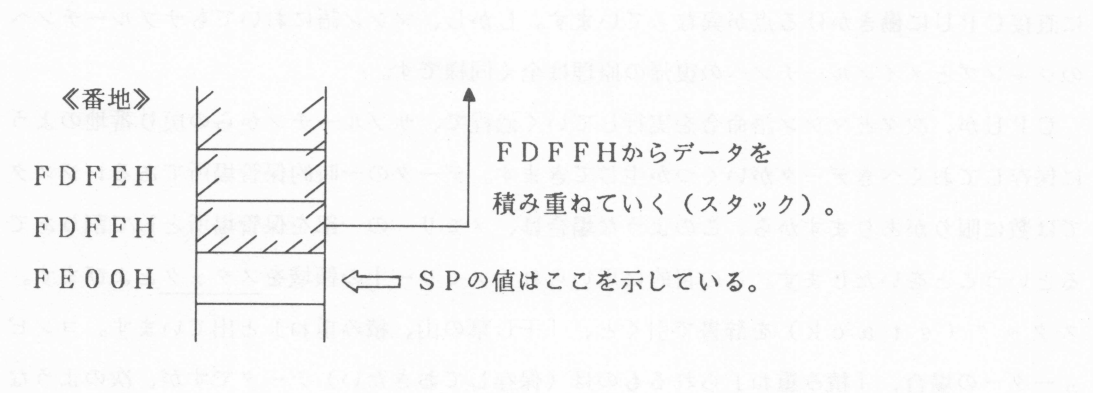



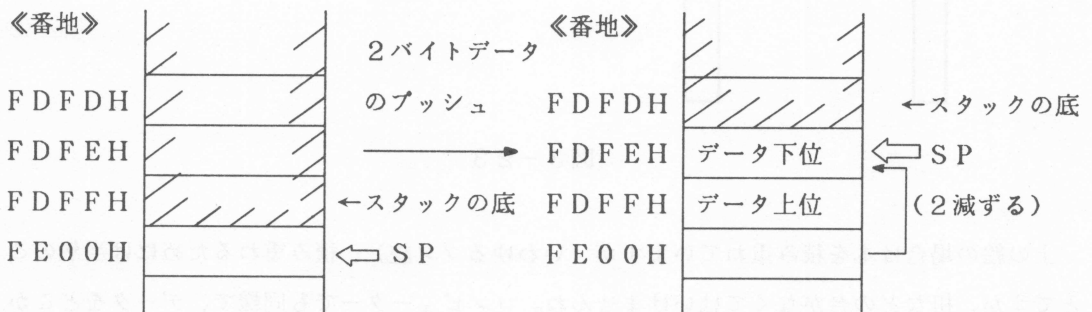
図３－２３

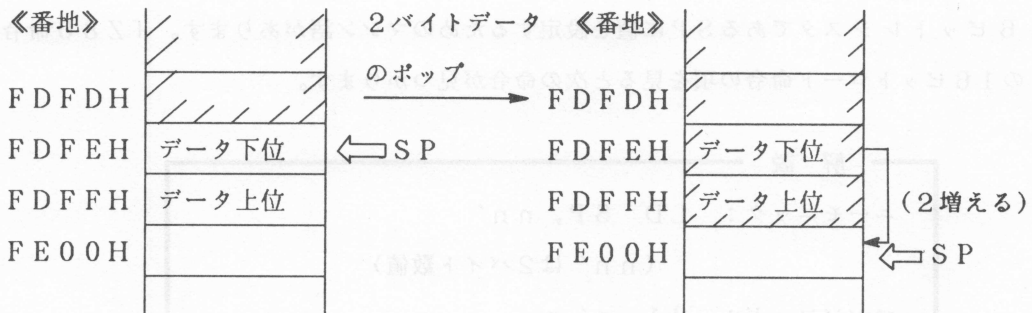
上の絵の場合は本を積み重ねていますが（いわゆるツン読）、積み重ねるためには当然のことですが、机などの台がなくてはいけませんね。コンピューターでも同様で、データをどこから積み重ねていけばよいのか　――　スタックがどこなのか？　――　をＣＰＵは知っていなくてはなりません。そのために設けられているレジスタがスタックポインタ（ＳＰ）です。

3－18／スタックポインタの働き

スタックポインタ（記号ＳＰ）は、１６ビットのレジスタで、スタックの位置を記憶しておくために用いられます。

たとえば、ＳＰに１６ビット数値　ＦＥ００Ｈ　がセットされていたとします。このときスタックは次のように設定されていることを意味します。

図の場合、スタックはＦＤＦＦＨ番地から番地の数字の若い方に向かって設けられています。一般に、ＳＰの示す値から１を引いた値が、スタックの一番“底”を示しています。データはこの番地から若い方に積み重ねられるのです。

では、スタックへのデータの積み重ね（プッシュ＝ｐｕｓｈ　という）について説明いたします。まず大切なことは、データは２バイトずつ積み重ねられる点です。従って、スタックへのデータのプッシュが行なわれると、ＳＰの値は自動的に２減ずるようにＣＰＵは設計されています。

上図を見ていただくと、ＳＰの働きがわかると思いますが、２バイトデータの積み重ねにおいても「上下位逆転」が生じることに注意して下さい。

逆に、ＣＰＵがスタックに積んでおいたデータを取り込む（ポップ＝ｐｏｐ　という）ときはどうでしょう。ＳＰの値は、積み重ねられたデータの一番「上」を示しているはずですから、ＣＰＵはＳＰの値を参照して、そこから２バイト分のデータを取り込めばよいのです。同時に、ＳＰの値は自動的に２増加するようになっています。

ＣＰＵが２バイトデータをポップした後も、積み重ねたデータはメモリーに残りますが、もはやＣＰＵは関知しません。そこはスタックの底と見なされますから、次に新しいデータをプッシュする時は、平気で書き換えてしまいます。

ＣＰＵとスタックとのデータのやりとり（プッシュとポップ）、それに伴うＳＰの動きは理解できましたか？　このことがわかると次の重要な点に気づくはずです。

スタックに積み重ねられているデータは大切なものであるから、これを理由なく破壊してはならない。そのためにも、ＳＰがどこを示しているか把握しておく必要がある。

私たちは、今まで気楽にモニターを起動しマシン語プログラムを入力・実行してきました。幸い今まではトラブルなく済んでいたことでしょう。しかし、もし偶然に、プログラムを格納する場所が、ＢＡＳＩＣインタプリターやモニターの使用するスタックと重なってしまったら

どうなるでしょう。スタックに保存されているデータは、それらのシステムプログラムにとって大切なデータであるはずですから、これを勝手に変更してしまうと、最悪の場合、ＢＡＳＩＣやモニターに戻った時、システムが壊れて「暴走」するかもしれませんね。

このような危険を避けるためにも、私たちはＢＡＳＩＣインタプリターやモニターが動いている時のＳＰの値について知識を持たねばなりません。

３－19／システムの使用するスタック

１６ビットレジスタであるＳＰに値を設定するためのマシン語があります。「Ｚ８０命令表」の１６ビットロード命令の項を見ると次の命令が見つかります。

解　説

ニーモニック：　ＬＤ　ＳＰ，ｎｎ′
（ｎｎ′は２バイト数値）

マシンコード：　３１　ｎ′ｎ

機　　能：　ＳＰ　←　ｎｎ′

Ｘ１の電源をＯＮし、ＢＡＳＩＣのシステムテープのロードが終了すると、ＣＰＵはまずシステム初期化ルーチン（００ＦＡＨ番地から始まっている）を実行します。このルーチン内では、

```
ＬＤ　ＳＰ，００００Ｈ
```

が実行されて、最初のスタックは　ＦＦＦＦＨ番地より若い方に向かって設定されます。試みに、モニターを起動し、　＊Ｄ　ＦＦＣ０　ＦＦＦＦ　によりメモリー内容をダンプして下さい。

```
MON
*D FFC0 FFFF
:FFC0=00 FF 00 7D 00 FF 00 7D /.テ.}.テ.}
:FFC8=00 FF 00 7D 00 FF 00 7D /.テ.}.テ.}
:FFD0=00 FF 00 7D 4D 0B C1 21 /.テ.}M.チ!
:FFD8=C0 23 08 08 12 08 CA 03 /タ#....ハ.
:FFE0=6D A2 26 07 EA 04 07 30 /m「&.■..3
:FFE8=07 04 07 20 11 07 C3 04 /.■...$.
:FFF0=C3 FF FF FF 08 04 BB 20 /テ.土テテテ..
:FFF8=BB 11 F8 FF A6 11 03 10 /ササレ.ヲ...
*■
```

図3－24

ダンプリストは例えば上図のようになっていると思います。ＦＦＤ４Ｈ番地あたりから以降にデータが格納されている様子がわかりますね。

さて、システムの初期化が終了すると、ＣＰＵは、起動時メッセージ（ＳＨＡＲＰ－Ｈｕ ＢＡＳＩＣ …）を表示し、９Ｄ５４Ｈ番地から　ＦＥＦＦＨ番地までのメモリー内容をクリア（００Ｈにする）した後、ＢＡＳＩＣインタプリターをスタートさせます。このとき、

LD SP, 0FE00H

が実行されます。すなわち、ＢＡＳＩＣインタプリターが使用するスタックは、最初　ＦＤＦＦＨ番地から若い方に向かって設定されることになります。これも見てみましょう。モニターを起動し、　＊Ｄ　ＦＤＣ０　ＦＥ００　によりメモリー内容をダンプして下さい。結果は例えば次のようになっているでしょう。

```
:FDC0=00 00 00 00 00 00 00 00 /........
:FDC8=00 00 00 00 00 00 00 00 /........
:FDD0=00 00 00 00 00 00 4D 0B /......M.
:FDD8=CD 21 55 03 08 08 12 08 /ﾍ!U.....
:FDE0=CA 03 76 A0 26 07 03 00 /ﾊ.v &...
:FDE8=C3 04 5B 9E 43 70 5C 9E /ﾃ.[╭Cp¥╭
:FDF0=54 9D 4C 6F DD 6E 44 FF /T'LoﾝnD〒
:FDF8=64 20 55 9D D3 15 FF FF /d U'ﾓ.〒〒
:FE00=00 00 00 00 00 00 00 00 /........
```

図3－25

ＦＤＦＦＨ番地から若い方へ向かってデータが積み重ねられている様子がわかりますね。これらのデータは特別な必要のない限り破壊するべきではありません。

次に、モニター起動時のスタックについて考えます。 ＭＯＮ を実行すると、ＢＡＳＩＣインタプリターは、必要なデータをスタックにプッシュした後、

ＬＤ ＳＰ，００００Ｈ

を実行し、ここに２バイトデータ １００３Ｈ をプッシュします。このデータは、モニターのコマンド待ち処理をするルーチンの開始アドレスです。先に、 ＊Ｄ ＦＦＣ０ ＦＦＦＦ によりメモリーダンプをした際、 ＦＦＦＥＨ番地と ＦＦＦＦ番地に格納されていた

という２バイトデータがこれです。上下位逆転して格納されていますね。従って、モニターコ

マンドレベルでのＳＰの値は、ＦＦＦＥＨ番地を示していることになります。

私たちが、モニターＧコマンドを実行する時は、このようなスタックの設定（ＦＦＦＤＨ番地から若い方へ向かって）になっている訳ですね。３－１７節で紹介した「マニュアル１７６ページ」の説明中、「スタックポインターは、ＦＦＦＥ（１６進）にあります。」という記述は以上のことを背景にしている訳です。よろしいですか？

こうして、システム（ＢＡＳＩＣインタプリター及びモニター）が使用するスタックの位置がわかりましたから、次の２つの問題を具体的に扱うことができます。

問題１	マシン語プログラムを安心して格納できる領域はどこか？
問題２	モニターのＧコマンドによるマシン語プログラムの実行のされ方はどのようなものか？

次節からこれらの問題を解決していくことにいたします。

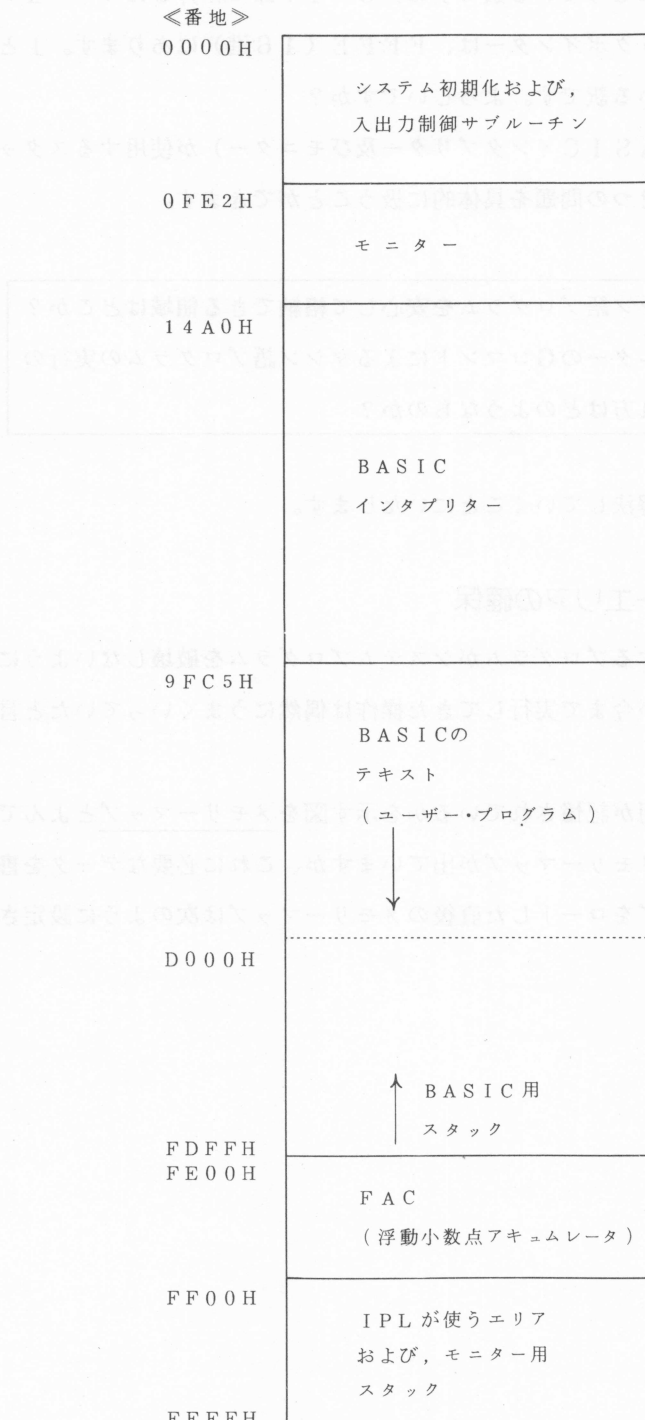
３－20／マシン語フリーエリアの確保

私たちがメモリーに格納するプログラムがシステムプログラムを破壊しないようにするという観点から見ると、私たちが今まで実行してきた操作は偶然にうまくいっていたと言わざるを得ません。

メモリー内のどの場所に何が記憶されているかを示す図をメモリーマップとよんでいます。マニュアル１８３ページにメモリーマップが出ていますが、これに必要なデータを書き加えてみました。ＢＡＳＩＣテープをロードした直後のメモリーマップは次のように設定されています。

図3-26

BASIC起動時のメモリーマップ

《番地》	
0000H	システム初期化および，入出力制御サブルーチン
0FE2H	モニター
14A0H	BASIC インタプリター
9FC5H	BASICのテキスト（ユーザー・プログラム）↓
D000H	↑ BASIC用スタック
FDFFH	
FE00H	FAC（浮動小数点アキュムレータ）
FF00H	IPLが使うエリアおよび，モニター用スタック
FFFFH	

このメモリーマップを見ていただくとわかるように、私たちが無意識に使用してきた　D0 00H番地のあたりは、上からはBASICのテキストエリアに迫られ、下からはBASICのスタックに迫られるサンドイッチになった領域であることがわかります。今までは、長いBASICプログラムを組まなかったことと、BASICスタックに多くのデータが積み重ねられなかったという「偶然」により、私たちのマシン語プログラムは正常に走ったのでした。

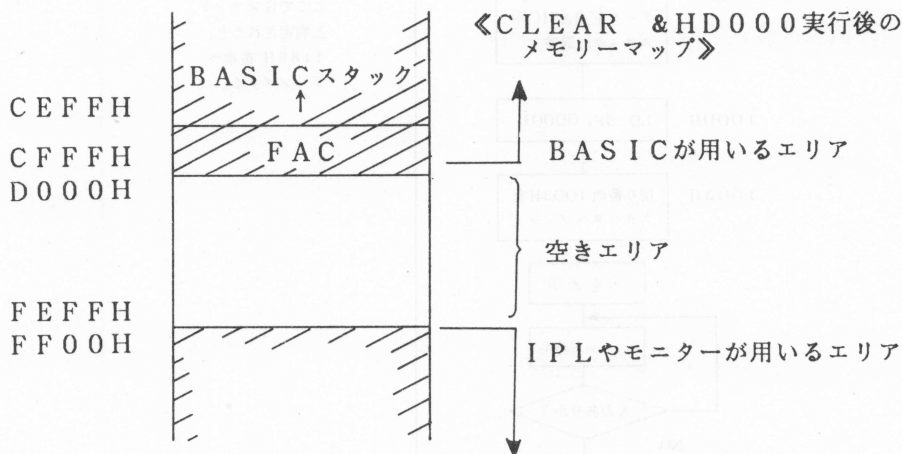
このような不安定な領域ではなく、もっと確実な領域に私たちユーザーのマシン語プログラムを格納するにはどうしたらよいのでしょう？　そのために用いられるBASICコマンドが

CLEAR

です。X1のBASICでは、このコマンドに同音異義が2つあります。ここで述べるのは、「変数・配列をすべてクリアする」ための　CLEAR　（マニュアル41ページ）ではなく、マニュアルの20ページにある　CLEAR　の方です。

このコマンドは次のような書式で用いられます。

CLEAR　アドレス（＝BASICが使用する上限ｱﾄﾞﾚｽ＋1）

たとえば、CLEAR　&HD000　を実行してみましょう。すると、以後BASICインタプリターは、メモリーのCFFFH番地より若い番地の方だけを使用するようになります。

今までＢＡＳＩＣ使用エリア内に含まれていた　Ｄ０００Ｈ～ＦＥＦＦＨ番地に「空き」が生じたことに注目して下さい。この部分こそ、私たちユーザーが安心してマシン語プログラムを格納しておけるフリーエリアなのです。

もし、フリーエリアをもっと広くとりたければ、たとえば　ＣＬＥＡＲ　＆ＨＣ０００　を実行すると、Ｃ０００Ｈ～ＦＥＦＦＨ番地をフリーエリアとして使えるようになります。ただし、ＢＡＳＩＣ使用エリアを余り狭くしすぎると、　Ｏｕｔ　ｏｆ　ｍｅｍｏｒｙ　エラーが出てしまいます。まあ普通は、　ＣＬＥＡＲ　＆ＨＤ０００　程度にしておくのが無難のようです。

以上のこと、理解されましたか？　今後はマシン語プログラムをメモリーに格納する際には、積極的に　ＣＬＥＡＲ　を活用して、ユーザー用マシン語フリーエリアを確保するよう努めましょう。こうして前節で提起された問題１は解決いたしました。

３－21／Ｇコマンドの解明

では最後に残った問題　――　モニターＧコマンドによるマシン語プログラムの実行のされ方　――　の解明に取りかかります。

モニターの処理を解読すると、おおよそ次の手順でＧコマンドを実行することがわかります。

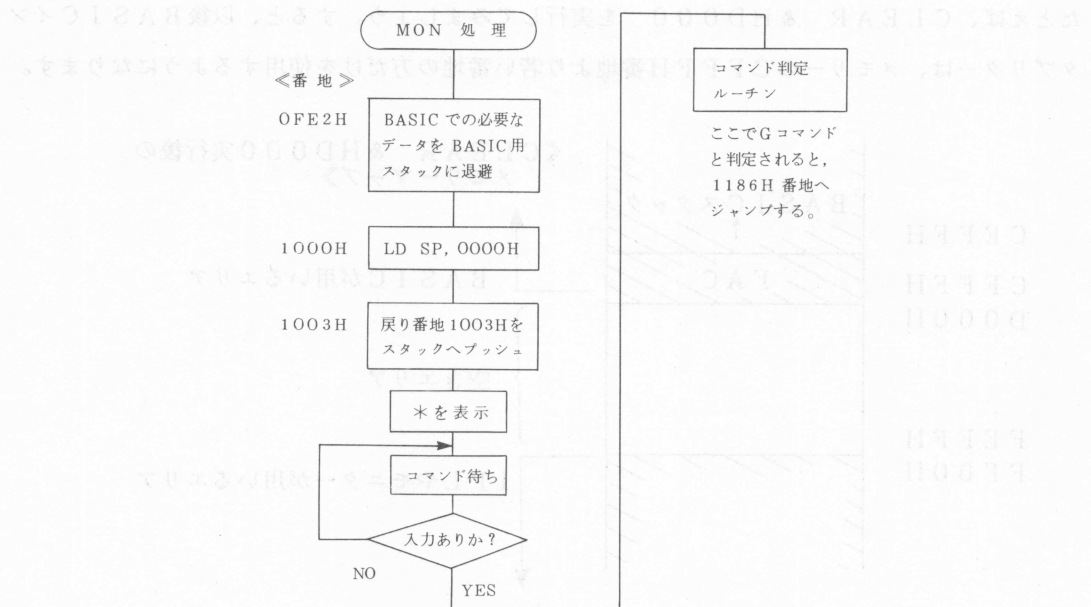
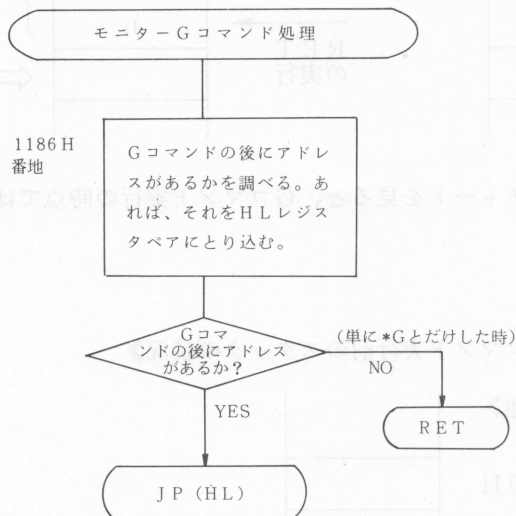



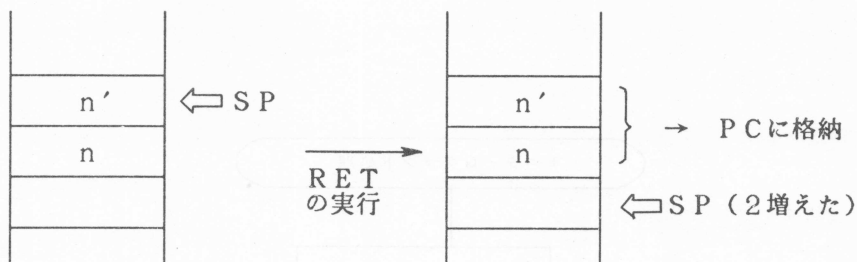
図3－27

Gコマンド処理ルーチンの２つの出口にあるマシン語が理解の鍵です。そろそろ、マシン語 RET の正体を述べる時が来たようです。

解　説

ニーモニック：　RET

マシンコード：　C9

機　　　能：　SPの示す番地、及びその次の番地から２バイト分をポップし、プログラムカウンタPCに格納する。

モニタールーチンのフローチャートを見ると、Gコマンド実行の時点では、スタックの状況は次のようになっています。

《Gコマンド実行前のスタックの状況》

＊G　アドレス　を実行すると、　1186H番地から始まる「Gコマンド処理ルーチン」へジャンプし、その中で、アドレスの値は2バイトデータとして、HLレジスタペアに格納されます。この場合、「Gコマンド処理ルーチン」の出口には、マシン語　JP　（HL）があります。

解　説

ニーモニック：　JP　（HL）

マシンコード：　E9

機　　能：　PC　←　HL　すなわち、HLレジスタペアの示すアドレスへジャンプする。

従って、私たちが今までよくやってきたように、　＊G　D000　を実行すると、「Gコ

マンド処理ルーチン」の出口で、 D000H番地へジャンプすることになります。すなわち、 D000H番地から格納されたマシン語プログラムが実行されます。

さて、私たちは「マシン語プログラムを停止させるおまじない」として、プログラムの最後にマシン語 RET を置いて来ました。なぜこれで、モニターのコマンドレベルに戻れるのかは、もうおわかりですね。

プログラム中で、最終的にSPの値を変更しない限り、プログラム末尾の RET の直前に来た時、SPの値は FFFEH番地になっているはずです。これで RET を実行すると、

《プログラム末尾における RET の実行》

となって、プログラムカウンタPCに 1003H がロードされ、従って、1003H番地から始まるモニターのコマンド待ちルーチンへジャンプできるのです。

3−22／マシン語サブルーチンの実行原理

前節で明らかになったモニターGコマンドの原理は、実は、サブルーチンの実行原理そのものなのです。私たちが作成し、 RET により停止させるマシン語プログラムは、モニタープログラムより呼び出される(コールされる)サブルーチンと見なされていることになります。従って、Gコマンドの名前が「ゴーサブ」なのですね。

本節では、私たちのマシン語プログラム中から、さらにサブルーチンを呼び出すためのマシン語を学びましょう。

解　説

ニーモニック：　ＣＡＬＬ　ｎｎ′
（ｎｎ′　は２バイト数値）

マシンコード：　ＣＤ　ｎ′ｎ　＊上下位逆転に注意。

機　　　能：　ＰＣの値（戻り番地）をスタックへプッシュした後、ＰＣ←ｎｎ′を行なう。すなわち、ｎｎ′番地からのサブルーチンを呼び出す。

たとえば次のような状況であるといたしましょう。

ＣＰＵが、　ＣＤ　００　Ｅ０　という３バイト命令を取り込むと、プログラムカウンタＰＣは次に実行すべきアドレスとして　Ｄ００３Ｈ番地を示します。ここで、ＣＰＵは　ＣＡＬＬ　０Ｅ０００Ｈ　を実行します。すなわち、ＰＣの現在値である２バイトデータ　Ｄ００３Ｈ　をスタックにプッシュします（それに伴いＳＰは２減じられます）。そして初めて、ＰＣに　Ｅ０００Ｈ　をロードします。こうして、　Ｅ０００Ｈ番地（ここからサブルーチンが始まる）へのジャンプが行なわれます。

さて、サブルーチンの最後は　ＲＥＴ　で終らなくてはなりません。ＣＰＵが　ＲＥＴ　を実行すると、ＳＰの示すアドレスから２バイト分（戻り番地）をＰＣへポップし（それに伴ってＳＰの値は２増える）、かくして、　ＣＡＬＬ　０Ｅ０００Ｈ　の次の番地　Ｄ００３Ｈに復帰できるのです。

以上がサブルーチンの実行原理なのです。スタックに戻り番地が格納されている点が重要です。ですから、私たちがサブルーチン中　ＬＤ　ＳＰ，ｎｎ′　などを実行して勝手にスタックの位置を変更してしまったら、どうなるかおわかりですね。戻り番地のデータが失われて、

ＲＥＴ　を実行しても戻れなくなってしまいます。最悪な場合は、「暴走」となる訳ですね。このように、ユーザープログラム内でのスタック操作にはくれぐれも注意が必要です（本書では、意図的にスタック操作を避ける方針をとっています）。

サブルーチン実行の実験をいたします。まず、　ＣＬＥＡＲ　＆ＨＤ０００　で、マシン語フリーエリアを確保して下さい。次に、　Ｄ０００Ｈ番地から次のプログラムを入力して下さい。

《サブルーチン・コールの実験》

アドレス	マシンコード	ニーモニック	コ　メ　ン　ト
Ｄ０００	ＣＤ０２１２	CALL 1202H	サブルーチンをコール
Ｄ００３	Ｃ９	RET	プログラムの停止

わずか４バイトのプログラムですからすぐ入力できますね。できましたら、　＊Ｇ　Ｄ０００　で実行して下さい。

結果は上のようになるはずですが、いかがですか？　実は、１２０２Ｈ番地から始まるサブルーチンは、ＨＬレジスタペアの内容を１６進４桁で表示するものです。私たちは前節で、＊Ｇ　Ｄ０００　を行なうと、ＨＬレジスタペアは　Ｄ０００Ｈ　に設定されることを学びましたね。この実験はそのことを証明しています。

ＨＬレジスタペアの内容を１６進４桁で画面表示させる処理は、かなり複雑そうですね。このように役立つサブルーチンが、モニターやＢＡＳＩＣインタプリターのシステムプログラム中にたくさん用意されています。これをシステムサブルーチンとよびます。システムサブルーチンを上手に利用すると、複雑な処理を少ないバイト数で実現できます（上がその例です！）。しかし、そのためにはモニターやＢＡＳＩＣインタプリターを解析して知識を得る必要があります。これは本書のページ数で説明することは不可能です。興味のある読者は自らの

手で、システムプログラムの解析に挑戦して下さい。そこはＺ８０のマシン語（とくにＸ１のマシン語）を勉強するための生きた教材そのものです。

3−23／第3章を終えるにあたって

３−１７節以降、私たちは「スタック」というものの意味と働きについて理解を深めてきました。現在の目でもう１度「画面反転プログラム」を見直してみましょう。

図３−２８　《画面反転サブルーチン》

アドレス	マシンコード	ニーモニック	
Ｄ０００	０１００２０	ＬＤ	ＢＣ，２０００Ｈ
Ｄ００３	ＥＤ７８	ＩＮ	Ａ，（Ｃ）
Ｄ００５	ＣＢＤＦ	ＳＥＴ	３，Ａ
Ｄ００７	ＥＤ７９	ＯＵＴ	（Ｃ），Ａ
Ｄ００９	０３	ＩＮＣ	ＢＣ
Ｄ００Ａ	７８	ＬＤ	Ａ，Ｂ
Ｄ００Ｂ	ＦＥ２３	ＣＰ	２３Ｈ
Ｄ００Ｄ	３８Ｆ４	ＪＲ	Ｃ，０Ｄ００３Ｈ
Ｄ００Ｆ	７９	ＬＤ	Ａ，Ｃ
Ｄ０１０	ＦＥＥ８	ＣＰ	０Ｅ８Ｈ
Ｄ０１２	３８ＥＦ	ＪＲ	Ｃ，０Ｄ００３Ｈ
Ｄ０１４	Ｃ９	ＲＥＴ	

いかがですか？　今の私たちは、これがマシン語サブルーチンであることを明確に理解できますね。しかも、それは相対アドレスによりリロケータブルになっていますから、任意の本体プログラムの中に組み込み、呼び出すことができますね。これらのことがわかるようになったことは、マシン語に対する私たちの実力向上を示しています！　自信を持って下さい！

次章からは、いよいよマシン語による高速ゲームの作成を始めます。画面反転サブルーチンも応用されます。楽しみにしていて下さい。

第4章 マシン語ゲームに挑戦！

■第4章 マシン語ゲームに挑戦！

4−1／ゲームの選定

第３章で、私たちはテレビ画面をマシン語で制御する方法をマスターしました。本章では、いよいよマシン語ゲームに挑戦です。

どんなゲームを採用するか考えたのですが、ブロック崩し型のゲームや、インベーダー型のゲームは、よく例として採り上げられているようですから、本書では、「追いかけっこ型」のゲームを材料にいたします。「追いかけっこ型」ゲームの代表は、「パックマン」や「平安京エイリアン」ですが、ここでは、映画『トロン』に題材をとり、「トロン・ゲーム」を考えてみましょう。

余談になりますが、『トロン』を以前見た時、「面白い題だな」と思っていました。それが、Ｘ１でＢＡＳＩＣを勉強してから、「ＴＲＯＮ」というコマンドに出会い、このコマンドを題名にしたのがわかりました。ついでながら、　ＴＲＯＮ　というのは、Ｔｒａｃｅ　Ｏｎの略ですね。これを実行すると、ＢＡＳＩＣインタプリターが今、行番号いくつの所を実行しているか追跡することができます。ＴＲＯＮモードからの脱出は、　ＴＲＯＦＦ　を実行すればよいのです。

さて、映画『トロン』では、レーザー分解装置により、コンピューター世界に入ってしまった主人公が、バイクに乗って追跡ゲームをさせられるシーンがありました（「ライトサイクルゲーム」というのだそうです）。この部分をＸ１でゲーム化してみましょう。

まず、ゲームの構成を把むために、オールＢＡＳＩＣで作成してみました。

4−2／オールBASIC版作成にあたって

オールＢＡＳＩＣ版の目的ですが、これはあくまで全体をつかむためのテスト用として作成します。従って、プログラムの構造を明確にすることが第１目的となります。

高速化は後に、マシン語版で考えることにし、速度を犠牲にしてでも、プログラムリストを見やすくすることに努めました。マルチステートメントはなるべく避け、ラベルと注釈（ＲＥＭないしは、その省略形の　）を多用しました。これらは、速度を遅くする大きな原因ですが、敢えてこういたします。

次にプログラムリストを掲げます。

図4－1　BASICリスト

```
10 ' ████████████████████████████████████████
20 ' ███                                  ███
30 ' ███     TRON GAME   ver.1            ███
40 ' ███                                  ███
50 ' ███          by  Yasuhiro Shimizu    ███
60 ' ███                                  ███
70 ' ███          1983.10.15              ███
80 ' ███                                  ███
90 ' ████████████████████████████████████████
100 '
110 ' █████      ショキカ      █████
120 '
130 INIT :WIDTH 40 :CLS 4 :CLICK OFF :TEMPO 200
140 DEFINT A-Z
150 '
160 GOSUB "オープニング"
170 GOSUB "キャラクター・テイギ"
180 '
190 ' █████      スウチ ノ ショキセッテイ      █████
200 '
210 DX=-1 :DY=0
220 LI=ASC("—") :B=120
230 '
240 DEF FNT(X,Y)=PEEK@(&H3000+X+Y*40)
250 DEF FNS(V,W)=PEEK@(&H3000+V+W*40)
260 '
270 ' █████      セツメイ      █████
280 '
290 CLS
300 LOCATE 10,0 :COLOR 4 :CSIZE 3 :PRINT#0 "TRON GAME" :CSIZE 0
310 LOCATE 13,4 :COLOR 6 :PRINT "TRON: "
320 LOCATE 18,4 :COLOR 7 :CGEN 1 :PRINT CHR$(120) :CGEN 0
330 LOCATE 13,6 :COLOR 2 :PRINT "SARK: "
340 LOCATE 18,6 :COLOR 7 :CGEN 1 :PRINT CHR$(220) :CGEN 0
350 LOCATE 16,8  :PRINT "KEY"
360 LOCATE 17,10  :PRINT "8"
370 LOCATE 16,11 :PRINT "4 6"
380 LOCATE 17,12  :PRINT "2"
390 LOCATE 13,14 :COLOR 3 :PRINT "10 POINT MATCH"
400 LOCATE 13,18 :COLOR 7 :PRINT "HIT "
410 LOCATE 17,18 :CFLASH 1 :PRINT "RETURN KEY" :CFLASH 0
420 REPEAT
430   I$=INKEY$
440   UNTIL I$=CHR$(13)
450 '
460 ' █████      ゲーム スタート      █████
470 '
480 T=0 :S=0 :G=0
490 CLS
500 LOCATE 8,8 :COLOR 5 :CSIZE 3
510 PRINT#0 "GAME START" :CSIZE 0
520 MUSIC "O4R2C3DEFGAB+C"
530 '
1000 ' █████      ガメン ヅクリ      █████
1010 '
1020 CONSOLE 1,24 :CLS :CONSOLE
1030 COLOR 4
1040 LINE ( 0, 1)-(38, 1),"—"
1050 LINE ( 1,23)-(38,23),"—"
1060 LINE ( 0, 2)-( 0,22),"|"
1070 LINE (39, 2)-(39,22),"|"
1080 LOCATE  0, 1 :PRINT "┌"
1090 LOCATE 39, 1 :PRINT "┐"
1100 LOCATE  0,23 :PRINT "└"
1110 LOCATE 39,23 :PRINT "┘";
1120 LOCATE  1, 0 :COLOR 6 :PRINT "TRON:";
1130 LOCATE 20, 0 :COLOR 2 :PRINT "SARK:";
1140 '
1150 ' █████      シュッパツ イチ ノ ケッテイ      █████
1160 '
1170 RANDOMIZE  TIME-20864
1180 X=INT(RND*20)+10
1190 Y=INT(RND*13)+6
1200 REPEAT
1210   U=INT(RND*36)+2
1220   V=INT(RND*19)+3
1230   UNTIL (X-U)*(X-U)+(Y-V)*(Y-V)>=25
1240 DU=INT(RND*2)*2-1 :DV=0
1250 CGEN 1 :COLOR 7
1260 LOCATE X,Y :PRINT#0 CHR$(120)
```

```
1270 LOCATE U,V :PRINT#0 CHR$(220) :CGEN 1
1280 FOR I=1 TO 3 :BEEP :NEXT
1290 '
2000 ' ■■■■■   ﾒｲﾝ･ﾙｰﾌﾟ   ■■■■■
2010 '
2020 M=DU :N=DV :F=1
2030 I=INT(RND*20)+1 :IF I<=4 THEN 2100
2040 '
2050 IF Y<V AND FNS(U,V-1)=ASC(" ") THEN DU= 0 :DV=-1 :GOTO 2150
2060 IF Y>V AND FNS(U,V+1)=ASC(" ") THEN DU= 0 :DV= 1 :GOTO 2150
2070 IF X<U AND FNS(U-1,V)=ASC(" ") THEN DU=-1 :DV= 0 :GOTO 2150
2080 IF X>U AND FNS(U+1,V)=ASC(" ") THEN DU= 1 :DV= 0 :GOTO 2150
2090 '
2100 IF FNS(U,V-1)=ASC(" ") THEN DU= 0 :DV=-1 :GOTO 2150
2110 IF FNS(U,V+1)=ASC(" ") THEN DU= 0 :DV= 1 :GOTO 2150
2120 IF FNS(U-1,V)=ASC(" ") THEN DU=-1 :DV= 0 :GOTO 2150
2130 IF FNS(U+1,V)=ASC(" ") THEN DU= 1 :DV= 0
2140 '
2150 IF DV=-1 THEN GOSUB 4000 :GOTO 2190
2160 IF DV=1  THEN GOSUB 4100 :GOTO 2190
2170 IF DU=-1 THEN GOSUB 4200 :GOTO 2190
2180 GOSUB 4300
2190 LOCATE U,V :COLOR 2 :CGEN 1
2200 PRINT CHR$(LI);
2210 U=U+DU :V=V+DV
2220 IF FNS(U,V)=ASC(" ") THEN 2280
2230 '
2240 LOCATE U,V :COLOR 2 :CGEN 0
2250 PRINT "X";
2260 T=T+1 :MUSIC "O4R3C1DEFGAB+CR3" :GOSUB "ｶﾁ ﾉ ﾊﾝﾃｲ" :IF H$<>"" THEN 3000  ELSE GOTO 1000
2270 '
2280 LOCATE U,V :COLOR 7
2290 PRINT CHR$(B) :CGEN 0
2300 '
2310 M=DX :N=DY :F=0
2320 I=STICK(0)
2330 IF I=8 THEN DX= 0 :DY=-1 :GOTO 2370
2340 IF I=2 THEN DX= 0 :DY= 1 :GOTO 2370
2350 IF I=4 THEN DX=-1 :DY= 0 :GOTO 2370
2360 IF I=6 THEN DX= 1 :DY= 0
2370 IF M*DX<0 OR N*DY<0 THEN DX=-DX :DY=-DY
2380 '
2390 IF DY=-1 THEN GOSUB 4000 :GOTO 2430
2400 IF DY= 1 THEN GOSUB 4100 :GOTO 2430
2410 IF DX=-1 THEN GOSUB 4200 :GOTO 2430
2420 GOSUB 4300
2430 LOCATE X,Y :COLOR 6 :CGEN 1
2440 PRINT CHR$(LI);
2450 X=X+DX :Y=Y+DY
2460 IF FNT(X,Y)=ASC(" ") THEN 2520
2470 '
2480 LOCATE X,Y :COLOR 6 :CGEN 0
2490 PRINT "X";
2500 S=S+1 :MUSIC "O4R3+C1BAGFEDCR3" :GOSUB "ｶﾁ ﾉ ﾊﾝﾃｲ" :IF H$<>"" THEN 3000 ELSE GOTO 1000
2510 '
2520 LOCATE X,Y :COLOR 7
2530 PRINT CHR$(B); :CGEN 0
2540 '
2550 GOTO 2000
2560 '
2570 '
3000 ' ■■■■■   ｹﾞｰﾑ ｵｰﾊﾞｰ   ■■■■■
3010 '
3020 FOR I=0 TO 1000
3030   A=PEEK@ (&H2000+I)
3040   A=(A OR &B1000)
3050   POKE@ &H2000+I,A
3060 NEXT
3070 '
3080 LOCATE 8,8 :COLOR 7 :CSIZE 2
3090 PRINT#0 " GAME!"+H$+" " :CSIZE 0
3100 IF H$="TRON" THEN MUSIC "O4C2DEFGAB+C"
3110 IF H$="SARK" THEN MUSIC "O4+C2BAGFEDC"
3120 '
3130 LOCATE 5,12 :PRINT "DO YOU WANT REPLAY [ Y or N ]"
3140 REPEAT
3150   I$=INKEY$
3160   UNTIL I$="Y" OR I$="N"
3170 '
3180 IF I$="Y" THEN 460     :' ------> ｹﾞｰﾑ ｽﾀｰﾄ
3190 INIT :CLS
3200 LOCATE 15,10 :PRINT "ｵﾂｶﾚｻﾏ ♥"
3210 END
3220 '
3980 ' ******   ｻﾌﾞﾙｰﾁﾝ   ******
3990 '
4000 ' ■■■■■   ｳｴ ﾆ ｲｸ   ■■■■■
4010 '
```

```
4020 LI=ASC("¦")
4030 IF M=-1 THEN LI=ASC("└")
4040 IF M= 1 THEN LI=ASC("┘")
4050 IF F=0 THEN B=100 ELSE B=200
4060 RETURN
4070 '
4100 '        ｼﾀ ﾆ ｲｸ
4110 '
4120 LI=ASC("|")
4130 IF M=-1 THEN LI=ASC("┌")
4140 IF M= 1 THEN LI=ASC("┐")
4150 IF F=0 THEN B=110 ELSE B=210
4160 RETURN
4170 '
4200 '        ﾋﾀﾞﾘ ﾆ ｲｸ
4210 '
4220 LI=ASC("─")
4230 IF N=-1 THEN LI=ASC("┐")
4240 IF N= 1 THEN LI=ASC("┘")
4250 IF F=0 THEN B=120 ELSE B=220
4260 RETURN
4270 '
4300 '        ﾐｷﾞ ﾆ ｲｸ
4310 '
4320 LI=ASC("─")
4330 IF N=-1 THEN LI=ASC("┌")
4340 IF N= 1 THEN LI=ASC("└")
4350 IF F=0 THEN B=130 ELSE B=230
4360 RETURN
4370 '
4380 '
4980 '
4990 '
5000   LABEL "ｶﾁ ﾉ ﾊﾝﾃｲ"
5010 '
5020 '
5030 '
5040 GOSUB "ｽｺｱ"
5050 IF G=1 THEN 5140
5060 '
5070 H$=""
5080 IF T=10 THEN H$="TRON"
5090 IF S=10 THEN H$="SARK"
5100 RETURN
5110 '
5120 ' ───────   ｼﾞｭｰｽ ﾉ ｼｮﾘ   ───────
5130 '
5140 H$=""
5150 IF T=S+2 THEN H$="TRON"
5160 IF S=T+2 THEN H$="SARK"
5170 RETURN
5180 '
5190 '
5980 '
5990 '
6000   LABEL "ｽｺｱ"
6010 '
6020 '
6030 '
6040 MUSIC "O5G1+C"
6050 LOCATE 8,0 :COLOR 6
6060 PRINTUSING "##";T;
6070 LOCATE 27,0 :COLOR 2
6080 PRINTUSING "##";S;
6090 '
6100 LOCATE 12,0 :PRINT "      ";
6110 IF T>=9 AND S>=9 AND T=S THEN G=1 :LOCATE 12,0 :COLOR 7 :CFLASH 1 :PRINT "ｼﾞｭｰｽ"; :CFLASH 0
 :FOR I=1 TO 2 :BEEP 1 :PAUSE 5 :BEEP 0 :PAUSE 5 :NEXT
6120 '
6130 RETURN
6140 '
6980 '
6990 '
7000   LABEL "ｵｰﾌﾟﾆﾝｸﾞ"
7010 '
7020 '
7030 '
7040 FOR I=1 TO 184
7050   COLOR (I MOD 6)+1 :PRINT "TRON ";
7060 NEXT
7070 PAUSE 20
7080 RETURN
7090 '
7100 '
7980 '
7990 '
8000   LABEL "ｷｬﾗｸﾀｰ･ﾃｲｷﾞ"
8010 '
```

```
8020 '
8030 '
8040 DEFCHR$(100)=HEXCHR$("0018183C3C181818667E7E3C3C7E7E66666666000066667E")
8050 DEFCHR$(110)=HEXCHR$("1818183C3C181800667E7E3C3C7E7E667E66660000666666")
8060 DEFCHR$(120)=HEXCHR$("000E7F1818000000000E7FFFFFE70000000000E7E7E70000")
8070 DEFCHR$(130)=HEXCHR$("0070FE181800000000070FEFFFFE70000000000E7E7E70000")
8080 '
8090 DEFCHR$(200)=HEXCHR$("0018183C3C18181866666600006666 7E000000000000000018")
8100 DEFCHR$(210)=HEXCHR$("1818183C3C1818007E6666000066666618000000000000000")
8110 DEFCHR$(220)=HEXCHR$("000E7F181800000000000E7E7E700000000000000000000000")
8120 DEFCHR$(230)=HEXCHR$("0070FE181800000000000E7E7E700000000000000000000000")
8130 '
8140 DEFCHR$(&H90)=HEXCHR$("000000FFFF00000000000000FFFF00000000000000FFFF000000")
8150 DEFCHR$(&H91)=HEXCHR$("181818181818181818181818181818181818181818181818")
8160 DEFCHR$(&H97)=HEXCHR$("000000F8F8181818000000F8F8181818000000F8F8181818")
8170 DEFCHR$(&H98)=HEXCHR$("181818F8F8000000181818F8F8000000181818F8F8000000")
8180 DEFCHR$(&H99)=HEXCHR$("1818181F1F0000001818181F1F0000001818181F1F000000")
8190 DEFCHR$(&H9A)=HEXCHR$("0000001F1F1818180000001F1F1818180000001F1F181818")
8200 '
8210 RETURN
8220 '
9000 ' ※※※※※※※※※※※※※※※※※※※※※※※※※※※※※※※※※※※※※※※※※※※※
9010 ' ※                                          ※
9020 ' ※    Sample Game for  X1 ﾏｼﾝｺﾞ ﾆｭｳﾓﾝ    ※
9030 ' ※                                          ※
9040 ' ※                  all BASIC version       ※
9050 ' ※                                          ※
9060 ' ※※※※※※※※※※※※※※※※※※※※※※※※※※※※※※※※※※※※※※※※※※※※
```

４−３／ゲーム作成の注意点

私としては、できる限り見やすいリストにしたつもりなので、詳細はリストを見ていただければわかると思いますが、いくつか注意すべき点がありますので以下に挙げます。

① ゲームをＲＵＮする前に、コンピューターの状態はどうなっているか不明なので、最初に必ず「初期化」を行ないましょう。Ｘ１のＢＡＳＩＣには、画面初期化のための便利なコマンド　ＩＮＩＴ　がありますので、これを使いました（１３０行）。

② １４０行は、すべての変数を整数型に宣言しています。ゲームでは、グラフィック画面でコンピューター・グラフィックスを行なう時以外は、ほとんど実数型変数は不要ですから、このようにしておくと、速度が上がります。

③ １６０行の「オープニング」について注意しておきます。よく経験するのは、ＲＵＮした後、画面に何の変化も起きない時間が数秒間続くケースです。これでは、プログラムがちゃんと動き始めたのか、何か原因不明のトラブルが生じたのかわかりませんね。特に、今の場合、１７０行で「ユーザーキャラクターの定義」をしていて、これは一般に、結構時間がかかるので、「オープニング」で素早く反応しておくことは大切だと思います。ただし、今の場合はユーザーキャラクターの数も少ないので、「オープニング」には「待ち」を入れておきます。

④ ２４０行と２５０行では、　ＦＮＴ（Ｘ，Ｙ）、　ＦＮＳ（Ｖ，Ｗ）　というユーザー用の関数を定義しています。トロン（操作する私たち）とサーク（コンピューター側）の移動を決定するためには、画面上の進みたい位置に障害物があるか否かを頻繁に判定しなくてはなりません。そこで、そのためのＶＲＡＭのＡＳＣＩＩコード読みとり関数を定義した訳です。

⑤ ３００行に、　ＰＲＩＮＴ＃０　とありますが、このコマンドは、倍文字やユーザー定義文字を表示するためのものです。

⑥ ４２０行〜４４０行は、リターンキー（ＡＳＣＩＩコード１３）のキー待ちです。

⑦ ４８０行は、ゲーム内での１サイクル開始のための数値初期化です。Ｔはトロンのスコア、Ｓはサークのスコアです。Ｇはジュース判定用の変数（フラグの働き！）で、０ならジュースでなく、１ならジュースを意味します。

⑧ １０２０行は、最上行にあるスコアを残して画面を消去するテクニックです。

⑨　１１７０行は、乱数系列を内蔵タイマーを利用して変える方法です。よくゲームの本に、ＲＡＮＤＯＭＩＺＥ　ＴＩＭＥ　としてあるのを見ますが、Ｘ１では注意が必要です。マニュアルを見るとわかりますが、　ＲＡＮＤＯＭＩＺＥ　の引数は、　－３２７６８～６５５３５　まで、内蔵タイマー変数　ＴＩＭＥ　の値は、　０～８６３９９　と変化しますね。ですから、６５５３５を越えた時の処理が必要です。　ＴＩＭＥ－２０８６４　としておくと、この値は、　－２０８６４～６５５３５　の範囲で変化して、ＲＡＮＤＯＭＩＺＥの引数範囲に納まり、Ｏｖｅｒｆｌｏｗ　エラーを生じません。

⑩　１２００～１２３０行は、サークの初期位置をトロンの初期位置から、距離５以上離すための処理です（ピタゴラスの定理を思い出して下さい！）。

⑪　１２８０行は、出発合図のベルです。

⑫　いよいよ、メイン・ループへ突入です。まず、２０２０行は、トロン・サーク共用のキャラクター選定ルーチン（４０００～４３６０行）へのデータ引き渡しの準備です。サブルーチンでは共用の変数として、ＭとＮを使用しています。Ｆは、トロン・サークの識別コードで０がトロン、１がサークです（トロン・サークフラグ！）。

⑬　２０３０行では、サークの行動に偶然性を持たせるための乱数設定をします。

⑭　２０５０～２０８０行は、サークがトロンに近づくための簡単なアルゴリズムです。

⑮　２１００～２１３０行は、サークがトロンと無関係に移動方向を決定するための部分です。トロンのいる方向に障害物がある時、および乱数が一定の値を示す時に、このルーチンが実行されます。

⑯　２１５０～２１８０行は、サークの軌跡およびバイクのキャラクターを選定しています。

⑰　２１９０～２２９０行は、サークのキャラクターを表示する部分です。障害物への衝突を判定し（２２２０行）、この時には、トロンのスコアを＋（プラス）して、音楽をならし、「勝負判定ルーチン」（５０００行）へ飛びます。

⑱　２３１０行は、２０２０行と同様に、サブルーチンへのデータ引き渡しです。トロン・サークフラグＦを０にして、トロンのモードにします。

⑲　２３２０～２３６０行は、キースキャンです。テンキーのどれが押されたかを判定します。

⑳　２３７０行は地味ですが重要な処理で、誤動作対策です。この処理の効果は、この行を消してゲームしてみるとすぐにわかります。

㉑　２３９０～２４２０行は、トロン軌跡およびバイクのキャラクター選定です。

㉒　２４３０～２５３０行は、トロンのキャラクター表示を行ないます。衝突の時は、サーク

のスコアを＋（プラス）し、音楽をならして、「勝負判定ルーチン」へ飛びます。

㉓ ２５５０行で、ループさせています。

㉔ ３０００～３２１０行は、ゲーム終了の処理で、「勝負判定ルーチン」から、ゲーム終了と判断されると、ここへ来ます。（終了判定は、Ｈ＄という文字変数が空でないことでなされる。） 特に、３０２０～３０６０行は、１画面反転処理です。この部分をマシン語化することはもうできますね！

㉕ ４０００～８２１０行には、サブルーチンをまとめておきました。少しでも速度を上げるために、移動方向によるキャラクター選定ルーチン（４０００～４３６０行）にはＬＡＢＥＬ文を使いませんでした。

本当は効果音など、もっと音楽を工夫すると一層面白くなるのですが、手もとに音楽のデータ（楽譜等）がないため、ドレミファソラシドでガマンしています。読者の皆様、是非工夫してみて下さい。また、サークがトロンを追跡するアルゴリズムも、もっと凝ったものを考えてみて下さい。

以上で、オールＢＡＳＩＣ版の説明を終えますが、これは、後のマシン語版の土台となりますので、是非入力して、遊んでみることをお勧めします。マシン語で少し大きめのプログラムを組むには、プログラムの構造がしっかり理解できていなくてはなりません。ＢＡＳＩＣプログラムのように「いきあたりばったり」的な作り方をすると大変に非能率的です。このオールＢＡＳＩＣ版を材料にして、プログラムの構造を頭に入れておいて下さい。また、オールＢＡＳＩＣ版は速度が遅いのがおわかりになると思いますが、プログラム中どこが速度を遅くしている原因であるかを考えてみて下さい。その部分こそ「マシン語化」のターゲットなのですから！

4－4／マシン語化にあたって

オールＢＡＳＩＣ版の「トロン・ゲーム」いかがでしたか？ トロンとサークの動きが、いかにもモタモタしていますね。本節からは、マシン語による高速化を考えてゆきましょう！

マシン語ゲームを作成するにあたり、基本的なことをまず考えておきます。オールマシン語版ができれば、もち論文句なしですが、後にマシン語プログラムを作成する段階になるとわかるように、ＢＡＳＩＣに比べて、マシン語によるプログラム作成の苦労は倍化いたします。また、ＢＡＳＩＣインタプリターが行なってくれていた「システムの初期化」―― 本格的な

周辺機器の制御　――　を自前でやらねばなりません。このようなことは、本書のページ数では納まらないことは明らかで、またＸ１のハードウェアの核心部分の理解も不可欠になります。そこで、オールマシン語（ＩＰＬ起動型とよく言う）のプログラムは断念いたします。

ではどうするかというと、中庸をとって、画面作りや、キャラクター定義など、ゲームの核心から離れた飾り的部分は、ＢＡＳＩＣで作成し、高速を要求されるメイン・ループの部分をマシン語サブルーチンにすることを考えます。このような作り方を、ＢＡＳＩＣとマシン語のリンクとよびます。

4－5／BASICとマシン語のリンクの実際

マシン語サブルーチンを、ＢＡＳＩＣ（のメインルーチン）とリンクさせる時には、マシン語部分の実行は、「ＢＡＳＩＣからマシン語サブルーチンを呼び出す」形で行なわれます。前章までのように、モニターを起動して、 ＊Ｇ　アドレス でないことは、もうおわかりですね。

マシン語サブルーチンの呼び出しは、当然　ＧＯＳＵＢ　では行なえません。ＧＯＳＵＢは行番号つきのＢＡＳＩＣサブルーチンの呼び出しのための命令でした。では、どうすればよいか？　困った時はマニュアルを熟読いたしましょう！

マニュアル１７８ページに、次のように書いてあります。

> ＢＡＳＩＣより機械語サブルーチンを呼び出す命令には、
> ＣＡＬＬ命令とＵＳＲ命令があります。

私たちは、この２つのＢＡＳＩＣ命令についてよく理解しておかねばなりません（次節から３節にわたり詳しく見ます）。

ここでは、もう少し基本的なことを考えておきましょう。ＢＡＳＩＣ部分と、マシン語部分が混在するのですから、「どこまでがＢＡＳＩＣで、どこからがマシン語」かを明確にしておく必要があります。すなわち、私たちは前章で確認したように、「マシン語フリーエリアの確保」を行なっておく必要があります。このためには、 ＣＬＥＡＲ　アドレス というＢＡＳＩＣ命令を用いるのでしたね。こうして、指定したアドレスから、ＦＥＦＦＨ番地までが、私たちのマシン語サブルーチンを格納しておく場所になるのです。

次に、そもそもマシン語サブルーチンはどのようにして書き込むのか？　について考えておきます。２つの方法があると思います。

第１の方法は、あらかじめモニターを起動して、前章までのようなやり方で、マシン語部分をメモリーに書き込んでおくというものです。

第２の方法は、ＢＡＳＩＣプログラムの中に、ＤＡＴＡ文でマシン語部分を書いておき、プログラム中、　ＲＥＡＤ　と　ＰＯＫＥ　により、メモリーに書き込む方法です。

本書では、マシン語部分の作成中は、テストも込めて第１の方法をとり、完成した後、第２の方法で１本のプログラムにまとめる方針でいきます。

以上の諸点よろしいですか？　ではお約束したように、「ＢＡＳＩＣからマシン語サブルーチンを呼び出す方法」から見ていきましょう。

4−6／BASICのCALL命令

ＢＡＳＩＣの　ＣＡＬＬ　命令は、 ＣＡＬＬ　マシン語サブルーチンの先頭アドレス という書式で用います。マシン語のニーモニックと同じ名称ですが、これはあくまでＢＡＳＩＣのコマンドとしてのＣＡＬＬですよ！

早速実験してみましょう。あらかじめ、 ＣＬＥＡＲ　＆ＨＤ０００ により、マシン語フリーエリア（Ｄ０００Ｈ〜ＦＥＦＦＨ番地）を確保しておいて下さい。マシン語サブルーチンの材料としては、前章で作成した「画面反転サブルーチン」を使いましょう。

ＭＯＮ　でモニターを起動し、マシン語をメモリーに書き込みます。フリーエリア内ならどこでも構いませんが、３−２３節に掲げたリストはＤ０００Ｈ番地からにしてありますから、まあこの番地からにしましょう。Ｄ０００Ｈ〜Ｄ０１４Ｈ番地に書き込み完了しましたか？

＊Ｒ　コマンドで、ＢＡＳＩＣの管理下に戻ります。では、いよいよ実行です！

```
CALL &HD000
```

を実行して下さい。予定通りの結果が出ましたか？　入力ミスがなければ、見事に白黒反転が起こるはずです。

こうして私たちは、ＢＡＳＩＣ　からマシン語サブルーチンを実行する方法を１つ身につけたことになります。

４−７／USR関数の理解をめざして−１−

次に２つ目の　ＵＳＲ関数　の説明をいたします。まず、マニュアル１７８ページを御覧下さい。かなり詳しい説明がありますね。ですから、マシン語について知識のある方でしたら、マニュアルの説明だけできっとおわかりになるでしょう。

しかし、マシン語を勉強し始めた当時の私には、この説明は難かしすぎました。実験しようにも、どうしたらよいかわかりませんでした。この体験、おそらく本書の読者であるマシン語初心者の方とも共有できると思います。そこで、本書では、２つの実験を用意いたしましたので、実験を通して理解を深めていただきたいと思います。

まず、ＣＡＬＬ命令と本質的に違う２点を確認しておきます。

確認１　：　ＵＳＲは、関数である。　ＣＡＬＬは、関数ではない。

確認２　：　ＣＡＬＬはいきなりで使えるが、ＵＳＲは実行に先立って
　　　　　　ＤＥＦＵＳＲ　で定義しておかねばならない。

関数については、おわかりですか？　代表的なものには、ＳＩＮがありますね。

これは、ＳＩＮ（数値又は数式）という形で用います。すると関数ですから、与えた数値又は数式のサイン（正弦）を計算して出力しますね。たとえば、　Ｙ＝ＳＩＮ（π／６）　とすると、変数Ｙに　π／６　ラジアンのサイン（＝０．５）が代入されます。

確認１は、ＵＳＲもこのような「関数」の一種だと言っているのです。従って、

Ｙ＝ＵＳＲ（データ）

のように用いると、データが加工されて、変数Ｙに代入されます。

今、「データが加工されて」と述べました。では、どのようにして加工されるのでしょう。ＳＩＮ関数の場合は、私たちは、入力した数値を加工する三角関数の法則を知っている訳です。では、ＵＳＲ関数の場合は、どのような法則なのでしょう。

これを理解する大枠が、確認２の内容なのです。すなわち、大ざっぱに言うと、

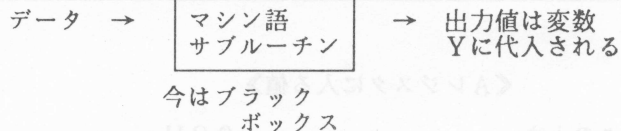
DEFUSR　であらかじめ指定しておいたアドレスから始まる
マシン語サブルーチンによって加工される。

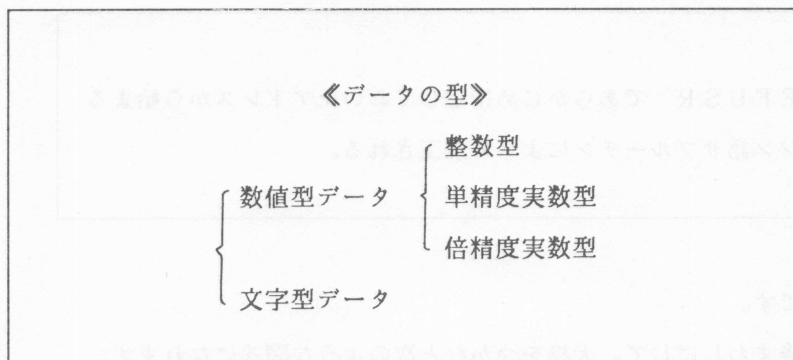
というのが答えです。

内部の詳細は後まわしにして、大枠をつかむと次のような図式になります。

DEFUSR　＝　マシン語サブルーチンの先頭アドレス
Y　＝　USR（データ）

データ　→　［マシン語サブルーチン］　→　出力値は変数Yに代入される

今はブラックボックス

まず大ざっぱに理解していただけましたか？　では次に、データを引き渡す仕組を見てまいりましょう。

入力するデータについて考察します。BASICでは、データの型を4種類に分けています。

これらのデータ（のいずれか）を、ＵＳＲ関数に入力いたしますと、まず、Ａレジスタにデータの型の識別コードが格納されます。

《Ａレジスタに入る値》

整数型データのとき	０２Ｈ
単精度実数型データのとき	０５Ｈ
倍精度実数型データのとき	０８Ｈ
文字型データのとき	０３Ｈ

まず、このことを実験で確認してゆきましょう。Ａレジスタの値は直接見ることはできませんから、メモリーに移して、ＰＥＥＫ関数で読むことにいたします。すなわち、マシン語サブルーチンの内容としては、Ａレジスタの内容をメモリー（ここでは仮にＥ０００Ｈ番地としておく）に移す作業を担当してもらいましょう。例によって、 ＣＬＥＡＲ　＆ＨＤ０００ でフリーエリアを確保し、仮に、Ｄ０００Ｈ番地からマシン語サブルーチンを書き込みます。

《ＵＳＲ関数のテスト１》

アドレス	マシンコード	ニーモニック
Ｄ０００	３２００Ｅ０	ＬＤ　　（０Ｅ０００Ｈ），Ａ
Ｄ００３	Ｃ９	ＲＥＴ

次に、テスト用のメインルーチンをＢＡＳＩＣで組みます。

図４－２　テストプログラム

```
10 REM ── USR ｶﾝｽｳ ﾄ A ﾚｼﾞｽﾀ ──
20 '
30 DEFUSR=&HD000
40 '
50 CLS
60 LOCATE 0,3
70 PRINT "ﾂｷﾞ ﾉ ﾄﾞﾚｶ ｦ ｼﾃｲｼﾃ､ ｸﾀﾞｻｲ｡ "
80 PRINT
90 PRINT
100 PRINT "ｾｲｽｳ ｶﾞﾀ ──────────> 1
110 PRINT "ﾀﾝｾｲﾄﾞ ｼﾞｯｽｳｶﾞﾀ ──> 2
120 PRINT "ﾊﾞｲｾｲﾄﾞ ｼﾞｯｽｳｶﾞﾀ ─> 3
130 PRINT "ﾓｼﾞｶﾞﾀ ────────────> 4
140 LOCATE 0,0 :BEEP :INPUT "USR ｶﾝｽｳ ﾆ ｱﾀｴﾙ ﾃﾞｰﾀ ﾉ ｼｭﾙｲ ﾊ ";N
150 LOCATE 0,13
160 ON N GOTO 170,180,190,230
170 BEEP :INPUT "ﾃﾞｰﾀ ﾊ ";A% :B=USR(A%) :GOTO 200
180 BEEP :INPUT "ﾃﾞｰﾀ ﾊ ";A! :B=USR(A!) :GOTO 200
190 BEEP :INPUT "ﾃﾞｰﾀ ﾊ ";A# :B=USR(A#)
200 AREG=PEEK(&HE000)
210 BEEP :PRINT "A ﾚｼﾞｽﾀ = ";RIGHT$("0"+HEX$(AREG),2)+"H","USR ｼｭﾂﾘｮｸ = ";B
220 END
230 BEEP :INPUT "ﾃﾞｰﾀ ﾊ ";A$ :B$=USR(A$)
240 AREG=PEEK(&HE000)
250 BEEP :PRINT "A ﾚｼﾞｽﾀ = ";RIGHT$("0"+HEX$(AREG),2)+"H","USR ｼｭﾂﾘｮｸ = ";B$
260 END
```

このＢＡＳＩＣプログラムを用いて、実験して下さい。実験例を下に掲げます。

［例］整数型データ　　１０　⇒　Ａレジスタ　＝　０２Ｈ
　　　　　　　　　　　　　　　　ＵＳＲ出力　＝　１０

単精度実数型データ　．５　⇒　Ａレジスタ　＝　０５Ｈ
　　　　　　　　　　　　　　　　ＵＳＲ出力　＝　．５

倍精度実数型データ　．000000005　⇒　Ａレジスタ　＝　０８Ｈ
　　　　　　　　　　　　　　　　ＵＳＲ出力　＝　５Ｅ－１０

文字型データ　“Ｘ”⇒　Ａレジスタ　＝　０３Ｈ
　　　　　　　　　　　　　　　　ＵＳＲ出力　＝　Ｘ（文字型のＸ）

以上で第１の実験を終えます。ＵＳＲ関数とＡレジスタの関係おわかりになりましたか？

４−８／USR関数の理解をめざして−２−

次にいよいよＵＳＲ関数の核心部分に迫ります。ＵＳＲ関数へ渡すデータは、どのように格納されるのか？　という点です。

これもデータの４つの型ですべて異なりますが、本書では、整数型データの場合のみを扱います。他の３つの型についてはマニュアルの説明と以下に掲げるプログラムを参考にして、皆さん自身で実験法を考えて下さい。

整数型データの場合は、ＨＬレジスタペアが、格納アドレスを示します。具体的には、整数値を２バイトのデータと見なして、

下位バイト	←　ＨＬで示されるアドレス
上位バイト	←　ＨＬで示されるアドレス＋１

と格納されます（上下位逆転していることに注意！）。　では実験にかかりましょう。今度のマシン語サブルーチンは、ＨＬレジスタペアの値、及び、ＨＬレジスタペアの示すアドレスの内容、ＨＬ＋１の示すアドレスの内容をメモリーの　Ｅ０００Ｈ番地から保存しておくものです（ＣＬＥＡＲ　＆ＨＤ０００　を忘れずに！）。

《ＵＳＲ関数のテスト２》

アドレス	マシンコード	ニーモニック
Ｄ０００	２２００Ｅ０	LD　(0E000H),HL
Ｄ００３	７Ｅ	LD　A,(HL)
Ｄ００４	３２０２Ｅ０	LD　(0E002H),A
Ｄ００７	２３	INC　HL
Ｄ００８	７Ｅ	LD　A,(HL)
Ｄ００９	３２０３Ｅ０	LD　(0E003H),A
Ｄ００Ｃ	Ｃ９	RET

このサブルーチンを実行すると、

メモリーのＥ０００Ｈ番地　←　Ｌレジスタの値

Ｅ００１Ｈ番地　←　Ｈレジスタの値

Ｅ００２Ｈ番地　←　ＨＬレジスタの示すメモリーの値

Ｅ００３Ｈ番地　←　ＨＬレジスタ＋１の示すメモリーの値

とデータが移動することはおわかりですね。（上下位逆転が起こっていますから注意！）

さて、テスト用のＢＡＳＩＣプログラムを掲げます。

図４－３

```
10 REM ─── USR カンスウ デ゛ノ セイスウ デ゛ータ ノ カクノウ ───
20 '
30 DEFUSR=&HD000
40 '
50 WIDTH 40 :CLS
60 BEEP :INPUT "セイスウ デ゛ータ = ";A%
70 B=USR(A%)
80 HREG=PEEK(&HE001) :H$=RIGHT$("0"+HEX$(HREG),2)
90 LREG=PEEK(&HE000) :L$=RIGHT$("0"+HEX$(LREG),2)
100 HLMEM=PEEK(&HE002) :ML$=RIGHT$("0"+HEX$(HLMEM),2)
110 HLNXT=PEEK(&HE003) :MH$=RIGHT$("0"+HEX$(HLNXT),2)
120 PRINT :BEEP
130 PRINT "HL レシ゛スタ = ";H$;L$;"H"
140 PRINT
150 PRINT "HL  ノ シメス メモリー ノ ナイヨウ = ";ML$;"H"
160 PRINT "HL+1 ノ シメス メモリー ノ ナイヨウ = ";MH$;"H"
170 PRINT
180 PRINT "USR シュツリョク = ";B
190 END
```

このプログラムを走らせ、いろいろな整数値を入れて、結果を見て下さい。ＵＳＲ関数によるデータ格納の様子、御理解いただけましたか？

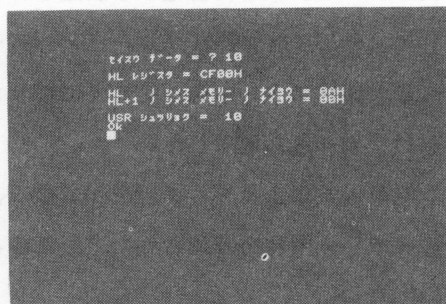



図４－４

以上、２つの実験で、ＵＳＲ関数により、マシン語サブルーチンへＢＡＳＩＣメインルーチンからデータを引き渡す方法がわかりました。マシン語サブルーチンでは、このようにして、

データの型及びデータを知ることができます。

さて、実験プログラムではいずれも、受け取ったデータ自体をサブルーチン内で加工することはしておりません。まあ、理屈っぽく言うと「何もしない」という加工をしていることになりますが、ともあれ、マシン語サブルーチンに入力されたデータをそのままメインルーチン側に返しています。ＵＳＲ関数の真髄を理解するには、本当はもう一歩進んで「加工・出力のされ方」をもっと調べる必要があるでしょう。しかし、本章の目的は、あくまでマシン語学習の一端としてのゲームの作成であり、また、本書では、マシン語サブルーチンから（加工されて）返ってくるデータを、ＢＡＳＩＣメインルーチンで利用しない予定ですので、ＵＳＲ関数による出力値の考察は、これ以上しないことにいたします。興味のある読者は是非、実験を試みて下さい。

［注］　２番目の実験に関して一言注意しておきます。　ＣＬＥＡＲ　＆ＨＤ０００　によりマシン語フリーエリアを確保してから実験しますと、ＨＬレジスタペアの示すアドレス値は、おそらく　ＣＦ００Ｈ　となっているはずです。実は今の場合、ＣＦ００Ｈ番地からＣＦＦＦＨ番地は、ＦＡＣ（Ｆｌｏａｔｉｎｇ－ｐｏｉｎｔ　Ａｃｃｕｍｕｌａｔｏｒ＝浮動小数点アキュムレータ）というエリアとして使用されています。実数データを扱うには、多くのバイト数を要しますが、ＣＰＵの内部レジスタには限りがありますから、メモリー上の一部を実数計算用のアキュムレータと見なして使います。これがＦＡＣです。マニュアル１８３ページのメモリーマップを見るとわかるように、ＦＡＣの開始アドレスは、（ＣＬＥＡＲ文で指定したアドレス）－１００Ｈ　と決められています。ですから、　ＣＬＥＡＲ　＆ＨＤ０００　のとき、ＦＡＣは　ＣＦ００Ｈ番地から設定されることになりますね。

ＵＳＲ（数値型データ）　により、マシン語サブルーチンへデータを引き渡す場合は、ＨＬレジスタペアは、このＦＡＣの開始アドレスを示すことになります。従って私たちの実験でも、ＨＬの値は　ＣＦ００Ｈ　となったのでした。

ところで、読者の中には　ＭＯＮ　によりモニターを起動して、ＣＦ００Ｈ番地と　ＣＦ０１Ｈ番地の２バイト分のメモリー内容を確認された方もあると思います。いかがでしたか？

多分、予期したデータ（入力した２バイト整数データ）はなかったはずです。しかし、この現象は実験のミスではありません。理由は、マシン語サブルーチンを実行して、ＦＡＣのデータを　Ｅ０００Ｈ番地以降へ退避させた後に、ＢＡＳＩＣインタプリターのシステムプ

ログラムがＦＡＣを書き換えてしまうからです。従って、ＦＡＣのデータを保存しておくには、このようにサブルーチン内の処理で退避させておく必要があるのです。

システムがＦＡＣを書き換えると述べましたが、もちろん書き換え方にも「法則」があるはずです。しかし、この部分は現段階では秘密のベールに包まれていますね。読者自らの手で、いつかこれらの秘密のベールをはいでみて下さい。

４－９／マシン語サブルーチンの配置

私たちは、２つのＢＡＳＩＣ命令（ＣＡＬＬとＵＳＲ）の基本的使い方をマスターすることで、ＢＡＳＩＣメインルーチンからマシン語サブルーチンを実行することができるようになりました。

もはや、私たちの前には、大きな目標 —— トロンゲームのマシン語化 —— だけが残されるのみとなりました。取り組む準備はよろしいですか？ では、マシン語ゲーム作成の始まり、始まり……。

まず、トロンゲームのオールＢＡＳＩＣ版のうち、どの部分をマシン語化するかを明確にしておきましょう。「メインループ」内を見て下さい。サークとトロンの行動決定・キャラクター選定のために、いかにもギコチナイ ＩＦ～ＴＨＥＮ 文の条件判断が続いています。これは明らかに速度低下の原因です。従って、行番号 ２０２０行～２２５０行、および２２８０行～２２９０行の部分を「サークの移動ルーチン」としてマシン語化します。同様に、２３１０行～２４９０行、および２５２０行～２５３０行を「トロンの移動ルーチン」としてマシン語に直します。これに伴い、４０００行～４３６０行のサブルーチンはマシン語ルーチンに吸収します。２２６０行と２５００行は、１サイクルの終了処理で「勝負判定」の部分ですから、ループの速度には大きく影響はないはずで、ＢＡＳＩＣのままにいたします。

また、「ゲームオーバー」ルーチン内の３０２０行～３０６０行は「画面反転」であり、ＢＡＳＩＣの ＦＯＲ～ＮＥＸＴ ループでは、いかにもモタモタしていますから、前章で私たちが完成した「画面反転サブルーチン」で置き換えましょう。

こうして、「マシン語化」の目標がはっきりいたしました。

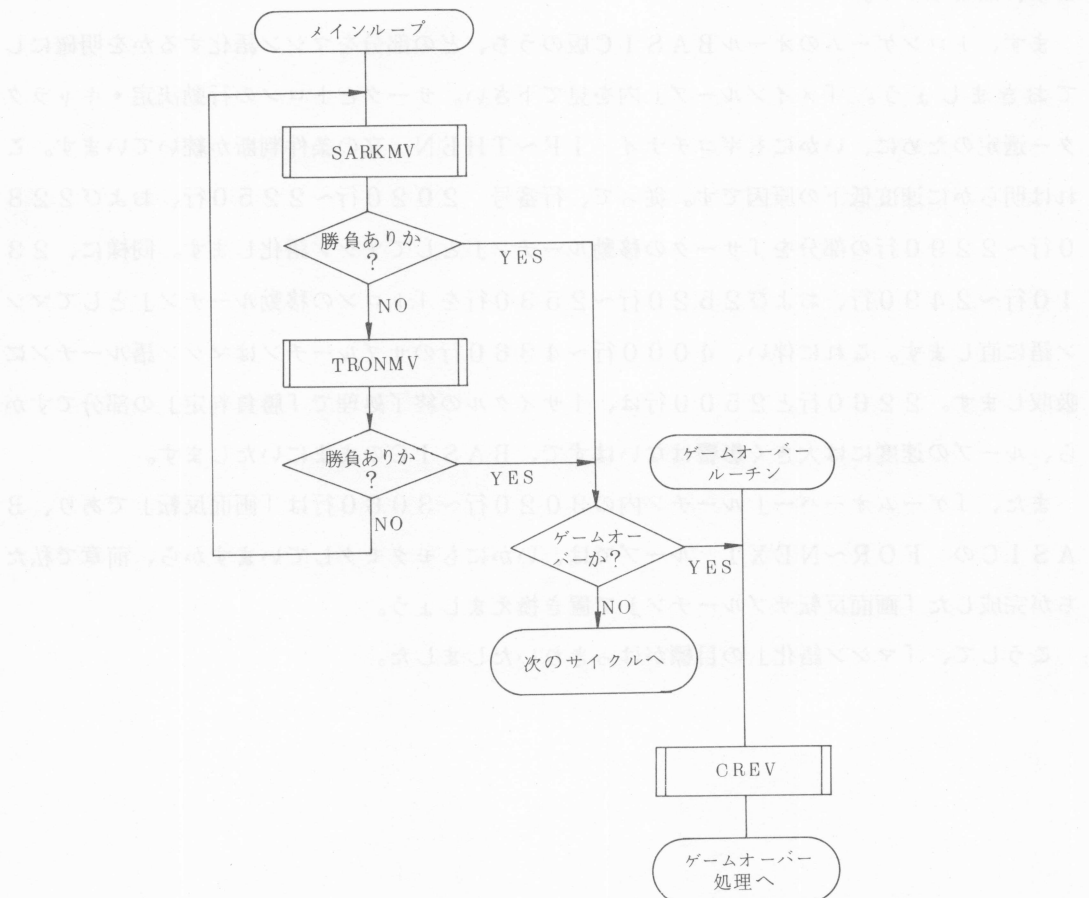
作成するマシン語サブルーチン

(1) サークの移動ルーチン
（ＳＡＲＫＭＶと名づける）

(2) トロンの移動ルーチン
（ＴＲＯＮＭＶと名づける）

(3) 画面反転ルーチン
（ＣＲＥＶと名づける）

メインループのフローチャートは次のようになります。各マシン語サブルーチン（ 　 で表示）が、どのように位置づけられているかを頭に入れて下さい。

図４－５

４－10／サブルーチンの仕様を決める!

次にする仕事は、３つのサブルーチンの仕様（どんな方法で、どんな順序でするか）の決定です。オールＢＡＳＩＣ版を参考に、各ルーチンでどのような処理がなされるべきかを思い描いてみて下さい。以下にフローチャートにして掲げます。

図４－６

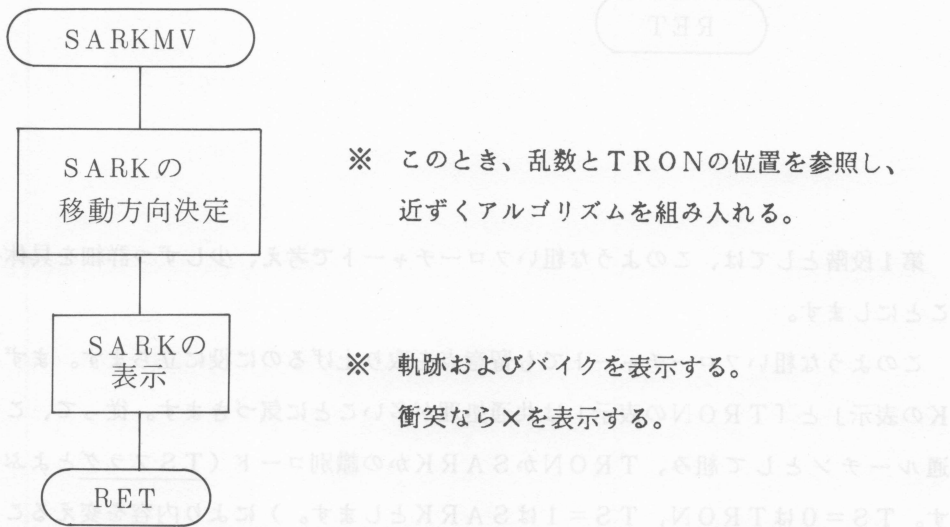



図４－７

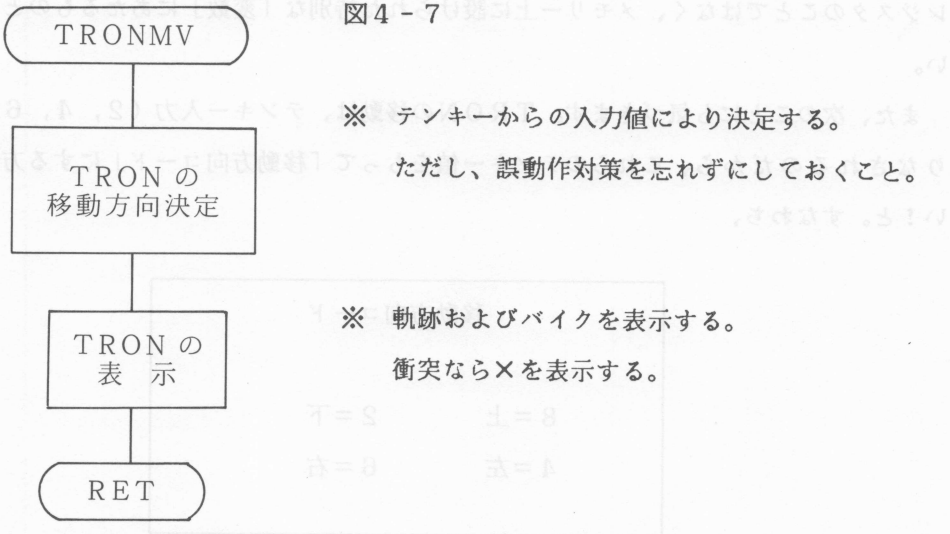
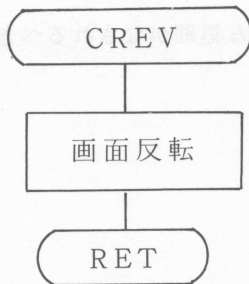



図4－8

※ すでに完成したサブルーチンを利用する。

第1段階としては、このような粗いフローチャートで考え、少しずつ詳細を具体化していくことにします。

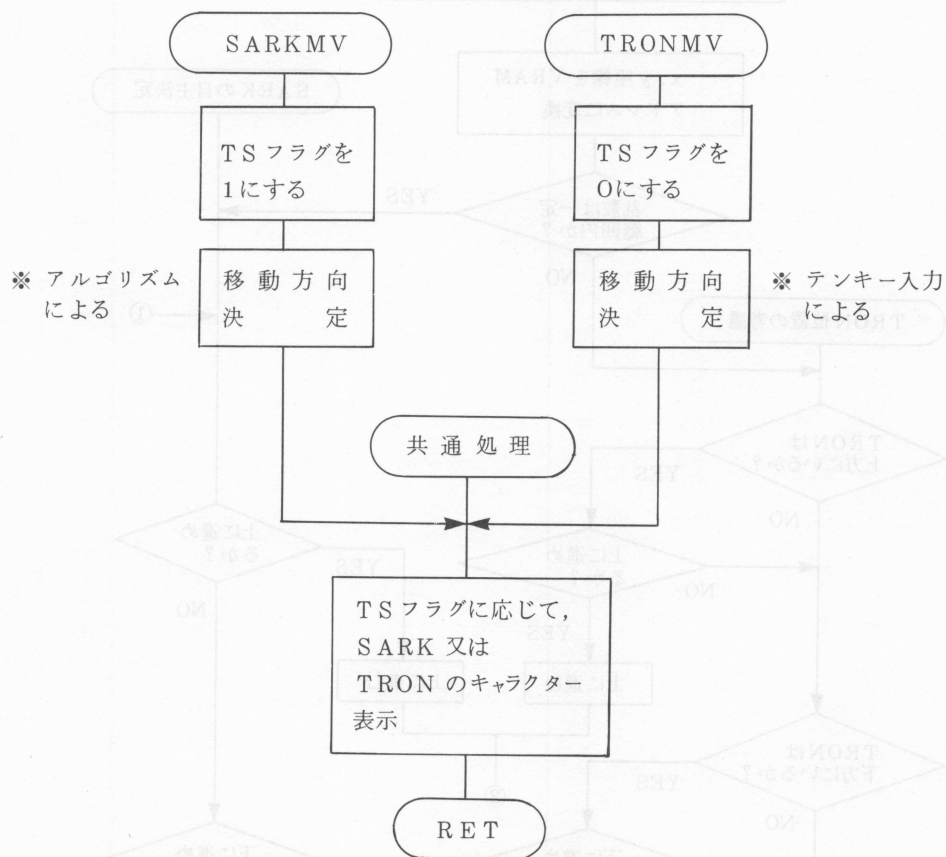
このような粗いフローチャートでも留意点を取り上げるのに役に立ちます。まず、「SARKの表示」と「TRONの表示」は共通処理が多いことに気づきます。従って、この部分は共通ルーチンとして組み、TRONかSARKかの識別コード（TSフラグとよぶことにします。TS＝0はTRON，TS＝1はSARKとします。）により内容を変えることにしましょう。この考えは、オールBASIC版でも利用しましたね。TSフラグの「フラグ」は、Fレジスタのことではなく、メモリー上に設けられた特別な「変数」にあたるものと考えて下さい。

また、次のことにも気づきます。TRONの移動は、テンキー入力（2，4，6，8）によりなされるのだから、これらのテンキー値をもって「移動方向コード」にする方が無駄がない！と。すなわち、

移動方向コード	
8＝上	2＝下
4＝左	6＝右

と決定します（この点はオールＢＡＳＩＣ版とは異なります）。

こうして、　ＳＡＲＫＭＶ　と　ＴＲＯＮＭＶ　という２つのルーチンは、次のように関連し合うことになりました。

図４－９

以上で、第１段階の作業が終了しました。だんだんイメージが明確になってきましたね。
（実は、オールＢＡＳＩＣ版もこのような構成になっているのです。）

４−11／移動方向決定ルーチンの具体化

次に移動方向決定のルーチンを具体化してゆきましょう。オールＢＡＳＩＣ版を参考に、フローチャートにします。

図４－１０

SARKの移動方向決定ルーチン
x，y座標をVRAM
アドレスに変換
乱数は一定
範囲内か？
YES
NO
SARKの自主決定
①
TRON位置の考慮
TRONは
上方にいるか？
YES
NO
上に進め
るか？
NO
YES
上に進む
上に進め
るか？
YES
NO
上に進む
②
TRONは
下方にいるか？
YES
NO
下に進め
るか？
NO
YES
下に進む
下に進め
るか？
YES
NO
下に進む
③
②
④

図4－10　続き

③
TRONは
左方にいるか？
YES
NO
左に進め
るか？
NO
YES
左に進む
左に進む
②
④
左に進め
るか？
YES
NO
TRONは
右方にいるか？
YES
NO
右に進め
るか？
NO
YES
右に進む
右に進む
右に進め
るか？
YES
NO
現在の方向に進む
衝突処理
①
SARKの
自主決定へ
②
表示ルーチンへ

図4－11

TRONの移動方向決定ルーチン
x，y座標をVRAMアドレスに変換
※ ここでテンキーの入力値をとり込んでおく。
誤動作対策
キースキャン
キーは8か？
YES
NO
上に進めるか？
YES
NO
衝突処理
上に進む
⑳
キーは2か？
YES
NO
下に進めるか？
YES
NO
衝突処理
下に進む
⑳
⑩
⑩
キーは4か？
YES
NO
左に進めるか？
YES
NO
衝突処理
左に進む
⑳
キーは6か？
YES
NO
右に進めるか？
YES
NO
衝突処理
右に進む
⑳
現在の方向に進めるか？
YES
NO
衝突処理
現在の方向に進む
⑳
表示ルーチンへ

以上のフローチャートにより、これらのルーチンから呼び出す下位のサブルーチンが浮かび上がってきます。いずれも、ＴＲＯＮ、ＳＡＲＫ共用といたします。

表４－１　下位サブルーチン

ラベル（名称）	主　要　な　内　容
ＡＤＣＡＬ	ｘ，ｙ座標をＶＲＡＭアドレスに変換する。
ＧＯＵＰ	上に進めるか否かを判断し、進めるならＣｙフラグを１にして上に進む。進めなければＣｙフラグを０にし、何もしない。［注］
ＧＯＵＰ１	無条件に上に進む。
ＧＯＤＮ	下に進めるか否かを判断し、進めるならＣｙフラグを１にして下に進む。進めなければＣｙフラグを０にし、何もしない。
ＧＯＤＮ１	無条件に下に進む。
ＧＯＬＴ	左に進めるか否かを判断し、進めるならＣｙフラグを１にして左に進む。進めなければＣｙフラグを０にし、何もしない。
ＧＯＬＴ１	無条件に左に進む。
ＧＯＲＴ	右に進めるか否かを判断し、進めるならＣｙフラグを１にして右に進む。進めなければＣｙフラグを０にし、何もしない。
ＧＯＲＴ１	無条件に右に進む。

［注］　ＧＯＵＰ，ＧＯＤＮ，ＧＯＬＴ，ＧＯＲＴ　の４つの条件判断つきルーチンでは、判断の結果、実際に進むのか、それとも進まないのかを、サブルーチンから戻った時に判定できなければなりません。そのため、ここでは、Ｃｙフラグ（キャリーフラグ）を用いてみました。　Ｃｙ＝１　なら、実際に進む処理をしたことを、また、　Ｃｙ＝０　なら、進む処理をしなかったことを意味します。

キャリーフラグを１にするには、次のマシン語を用います。

解　説

ニーモニック：　ＳＣＦ　［Ｓｅｔ　Ｃａｒｒｙ　Ｆｌａｇの略］
マシンコード：　３７
機　　　　能：　Ｃｙフラグをセットする。

では逆に、キャリーフラグを０にするには？　ＲＥＳＣＦ　のような命令があればよいので

すが、残念ながら「Ｚ８０命令表」にはありません。しかし、「キャリーフラグを逆転する命令」　ＣＣＦ　があるので、　ＳＣＦ　の後に、　ＣＣＦ　をすれば、結果として、Ｃｙは０になります。

ここでは別の方法でＣｙを０にしてみます。論理演算の利用です。

解説

ニーモニック：　ＯＲ Ａ

マシンコード：　Ｂ７

機　　能：　ＡレジスタとＡレジスタの論理和（ＯＲ）をとり、Ａレジスタに格納する。
次のフラグ変化をする。
●Ｃｙフラグ：つねにリセットされる。
●Ｚフラグ：Ａレジスタの内容が０ならセットされ、０以外ならリセットされる。

論理和というと恐ろしげですが、要するに対応するビット毎に次の規則を適用するだけです。

《論理和の規則》

１，１　⇒　１

１，０　⇒　１

０，１　⇒　１

０，０　⇒　０

たとえば、１０１０１０１１（２進表示）と、１１００１１００（２進表示）の論理和をとると、

```
       10101011
論理和) 11001100
       ────────
       11101111　（2進表示）
```

となります。さて、　ＯＲ　Ａ　という時には、Ａレジスタと Ａレジスタの論理和で、どのビットも同じですから、 １，１⇒１，　０，０⇒０ の規則から、Ａレジスタの内容は不変です。ただし、フラグの変化が重要です。ですから、この命令は、「Ｃｙフラグのリセット」や、「Ａレジスタの内容が０か否かを判定する」のによく用いられます。

4－12／キャラクター表示ルーチンの具体化

最後に、ＴＲＯＮ、ＳＡＲＫ共用のキャラクター表示ルーチンを具体化しておきましょう。

キャラクターのＡＳＣＩＩコードを確認しておきます（ユーザー定義文字として使用）。

表４－２

バイク・キャラクター		ＡＳＣＩＩコード	（１０進）
ＴＲＯＮ	上向き	６４Ｈ	１００
	下向き	６ＥＨ	１１０
	左向き	７８Ｈ	１２０
	右向き	８２Ｈ	１３０
ＳＡＲＫ	上向き	Ｃ８Ｈ	２００
	下向き	Ｄ２Ｈ	２１０
	左向き	ＤＣＨ	２２０
	右向き	Ｅ６Ｈ	２３０

軌跡キャラクター	ＡＳＣＩＩコード
─	９０Ｈ
│	９１Ｈ
┐	９７Ｈ
┘	９８Ｈ
└	９９Ｈ
┌	９ＡＨ

衝突キャラクター（標準モード）	ＡＳＣＩＩコード
×	Ｅ８Ｈ

フローチャートは以下の通りです。

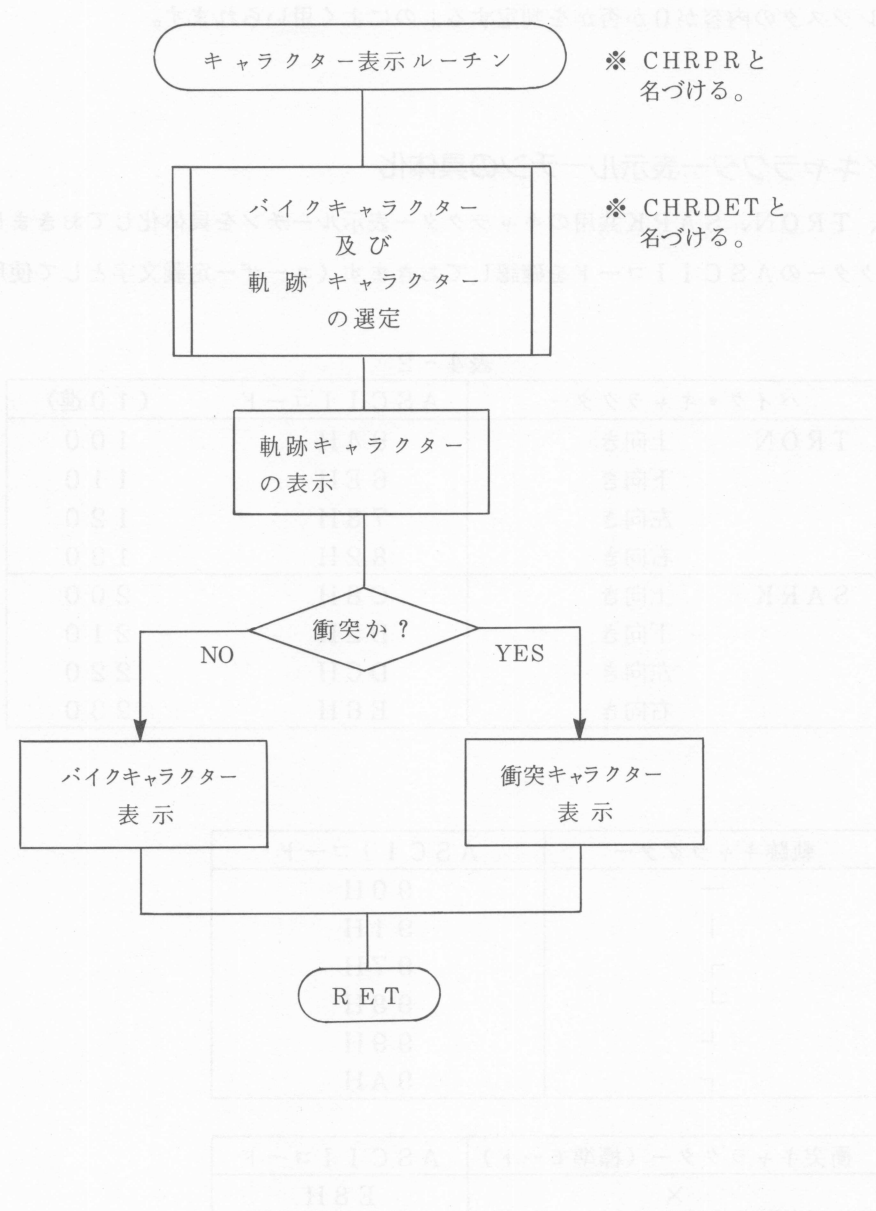
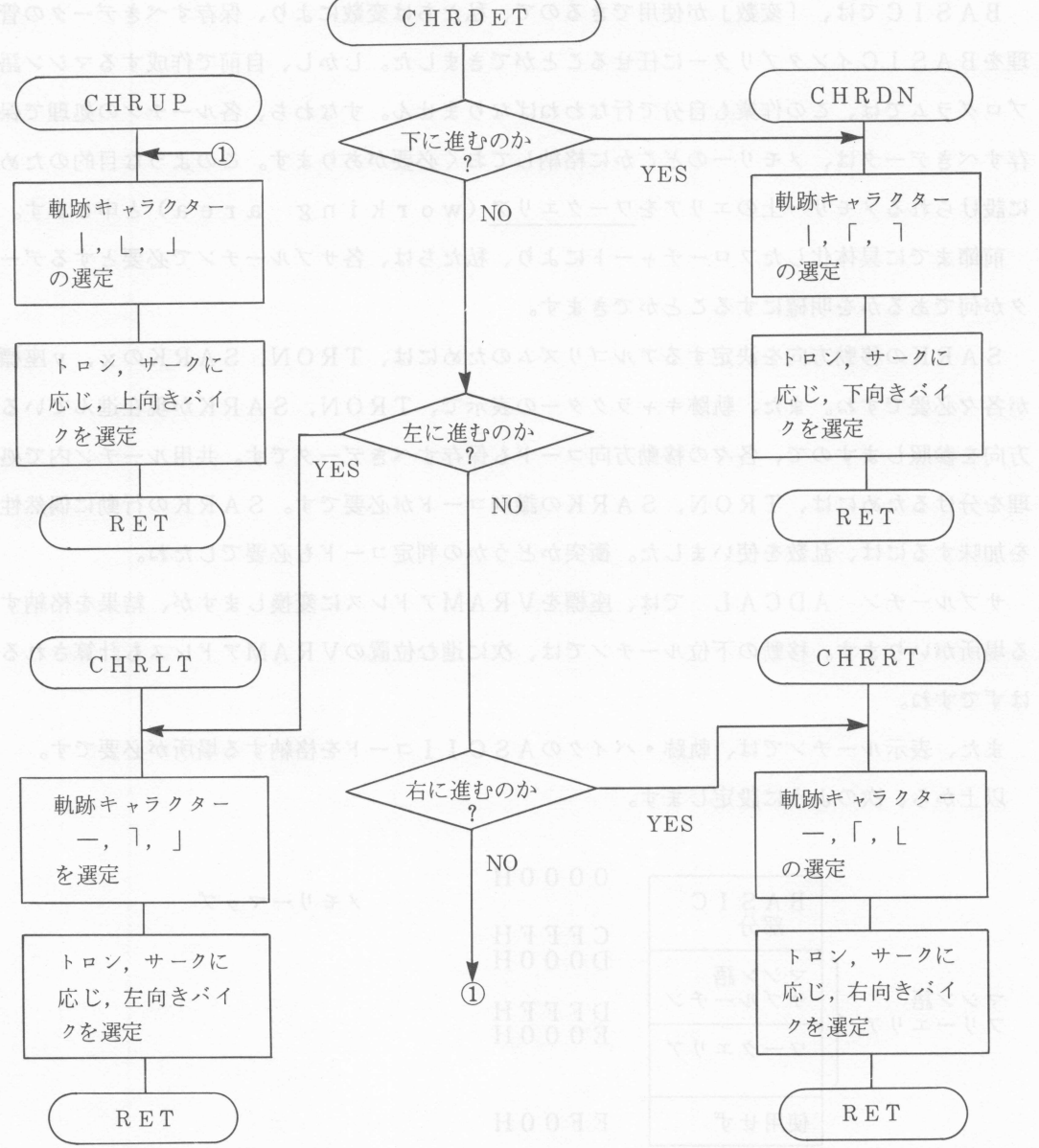
図4－12

キャラクター選定サブルーチン（CHRDET）は、次のように具体化します。

図4－13

［注］　4つの下位ルーチン　CHRUP，CHRDN，CHRLT，CHRRT　では、現在の方向コードと、新しい移動方向コードを比較して、軌跡キャラクターを選ぶことにします（オールBASIC版でも、この原理を用いました）。

４−13／ワークエリアの設定

ＢＡＳＩＣでは、「変数」が使用できるので、私たちは変数により、保存すべきデータの管理をＢＡＳＩＣインタプリターに任せることができました。しかし、自前で作成するマシン語プログラムでは、この作業も自分で行なわねばなりません。すなわち、各ルーチンの処理で保存すべきデータは、メモリーのどこかに格納しておく必要があります。このような目的のために設けられるメモリー上のエリアをワークエリア（ｗｏｒｋｉｎｇ　ａｒｅａ）と申します。

前節までに具体化したフローチャートにより、私たちは、各サブルーチンで必要とするデータが何であるかを明確にすることができます。

ＳＡＲＫの移動方向を決定するアルゴリズムのためには、ＴＲＯＮ，ＳＡＲＫのｘ，ｙ座標が各々必要ですね。また、軌跡キャラクターの表示で、ＴＲＯＮ，ＳＡＲＫが現在進んでいる方向を参照しますので、各々の移動方向コードも保存すべきデータです。共用ルーチン内で処理を分けるためには、ＴＲＯＮ，ＳＡＲＫの識別コードが必要です。ＳＡＲＫの行動に偶然性を加味するには、乱数を使いました。衝突かどうかの判定コードも必要でしたね。

サブルーチン　ＡＤＣＡＬ　では、座標をＶＲＡＭアドレスに変換しますが、結果を格納する場所がいります。移動の下位ルーチンでは、次に進む位置のＶＲＡＭアドレスも計算されるはずですね。

また、表示ルーチンでは、軌跡・バイクのＡＳＣＩＩコードを格納する場所が必要です。

以上から、次のように設定します。

このように各部分をメモリー上のどこに配置するかを示した図を　メモリーマップ　とよぶのでしたね。マシン語プログラムを作成する時は、必ずメモリーマップを明確にしておきましょう。

またワークエリアの詳細は次のように決めます。

表４－３　ワークエリア

アドレス	内　　　　容	ラベル（名称）
Ｅ０００Ｈ	ＴＲＯＮ　ｘ座標	ＴＲＸ
Ｅ００１Ｈ	ＴＲＯＮ　ｙ座標	ＴＲＹ
Ｅ００２Ｈ	ＳＡＲＫ　ｘ座標	ＳＫＸ
Ｅ００３Ｈ	ＳＡＲＫ　ｙ座標	ＳＫＹ
Ｅ００４Ｈ	ＴＲＯＮ　移動方向コード（２，４，６，８）	ＴＲＭＶ
Ｅ００５Ｈ	ＳＡＲＫ　移動方向コード	ＳＫＭＶ
Ｅ００６Ｈ	乱数	ＲＮＤ
Ｅ００７Ｈ	続行フラグ（０＝続行、１＝終了）	ＦＬＡＧ
Ｅ００８Ｈ	ＶＲＡＭ　現アドレス　下位バイト	ＡＤＲ
Ｅ００９Ｈ	ＶＲＡＭ　現アドレス　上位バイト	
Ｅ００ＡＨ	ＶＲＡＭ　新アドレス　下位バイト	ＮＡＤＲ
Ｅ００ＢＨ	ＶＲＡＭ　新アドレス　上位バイト	
Ｅ００ＣＨ	（計算用）移動方向コード	ＭＯＶＥ
Ｅ００ＤＨ	ＴＲＯＮ，ＳＡＲＫフラグ(0=ＴＲＯＮ，1=ＳＡＲＫ)	ＴＳ
Ｅ００ＥＨ	軌跡キャラクターのＡＳＣＩＩコード	ＯＲＢＩＴ
Ｅ００ＦＨ	バイクキャラクターのＡＳＣＩＩコード	ＢＩＫＥ

ワークエリアの各アドレスには、引用しやすいように、ラベル（名前）をつけておきます。（アドレスのように２バイトの場合は、上下位逆転が起きています。この時は、下位バイトの方のみにラベルをつけました。）

４－14／データ引き渡しの仕様

ＢＡＳＩＣの「変数」にあたるワークエリアの仕様がはっきりしましたので、次に各ルーチンが、どのようにワークエリアを書き換えてゆくかを明確に決めます。

この問題は、サブルーチン間のデータ引き渡しについての取り決めとも言えます。ＢＡＳＩＣインタプリター内には、多くのマシン語サブルーチンがありますが、これらの間でのデータ引き渡しは、レジスタを通じて行なわれることが多いようです。この方法は、省メモリー・高速化を図れますが、レジスタは数に限りがあるため、ＣＰＵの動作に伴い頻繁に値が変わります。従って、このような方法でシステムを組む時は、各サブルーチン内でのレジスタの値の動きを追跡し、保存されるレジスタ、書き換えられるレジスタをきちんと把んでおかねばなりま

せん。これは、初心者にとってはなかなか大変な仕事なのです（私もそうでした）。

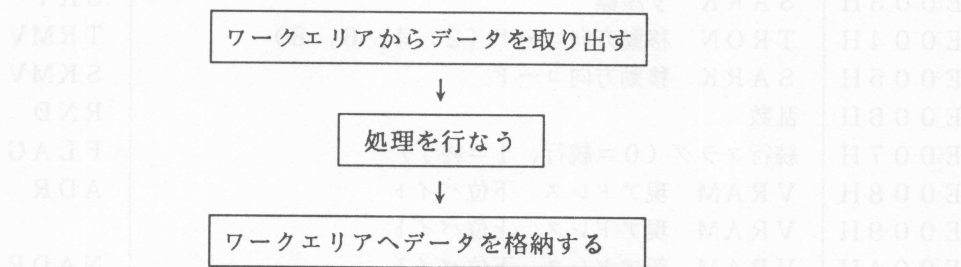
そこで本書のゲーム作成では、速度が多少遅くなり、メモリーが余計に使われるのをガマンしてでも、特定のデータを格納しておくメモリーのエリア（ワークエリア）を経由して、サブルーチン間のデータ引き渡しを行なう方法をとることにします。これにより、各サブルーチンでのデータの動きは次図のようになり、単純でわかりやすいプログラムにできます。

以下、ワークエリアの動きをサブルーチン毎に決めてゆきます。［アドレスはラベルで記します。］

表４－４　《下位サブルーチン》

名　称	参照するアドレス	書き換えるアドレス
ＡＤＣＡＬ	ＴＳ トロンのとき テンキー入力 （ＨＬレジスタペアの示すアドレス） ＴＲＸ，ＴＲＹ サークのとき ＳＫＭＶ ＳＫＸ，ＳＫＹ	ＭＯＶＥ ＡＤＲ（２バイト）
ＧＯＵＰ１ （上へ進む）	ＡＤＲ（２バイト） ＴＳ トロンのとき、ＴＲＹ サークのとき、ＳＫＹ	ＮＡＤＲ（２バイト） ＭＯＶＥ（←８） ＴＲＹ（１減ずる） ＳＫＹ（１減ずる）

GODN1 (下へ進む)	ADR(2バイト) TS トロンのとき、TRY サークのとき、SKY	NADR(2バイト) MOVE(←2) TRY(1増やす) SKY(1増やす)
GOLT1 (左へ進む)	ADR(2バイト) TS トロンのとき、TRX サークのとき、SKX	NADR(2バイト) MOVE(←4) TRX(1減ずる) SKX(1減ずる)
GORT1 (右へ進む)	ADR(2バイト) TS トロンのとき、TRX サークのとき、SKX	NADR(2バイト) MOVE(←6) TRX(1増やす) SKX(1増やす)
CHRDET (キャラクター選定)	MOVE TS トロンのとき、TRMV サークのとき、SKMV	ORBIT BIKE トロンのとき、TRMV サークのとき、SKMV

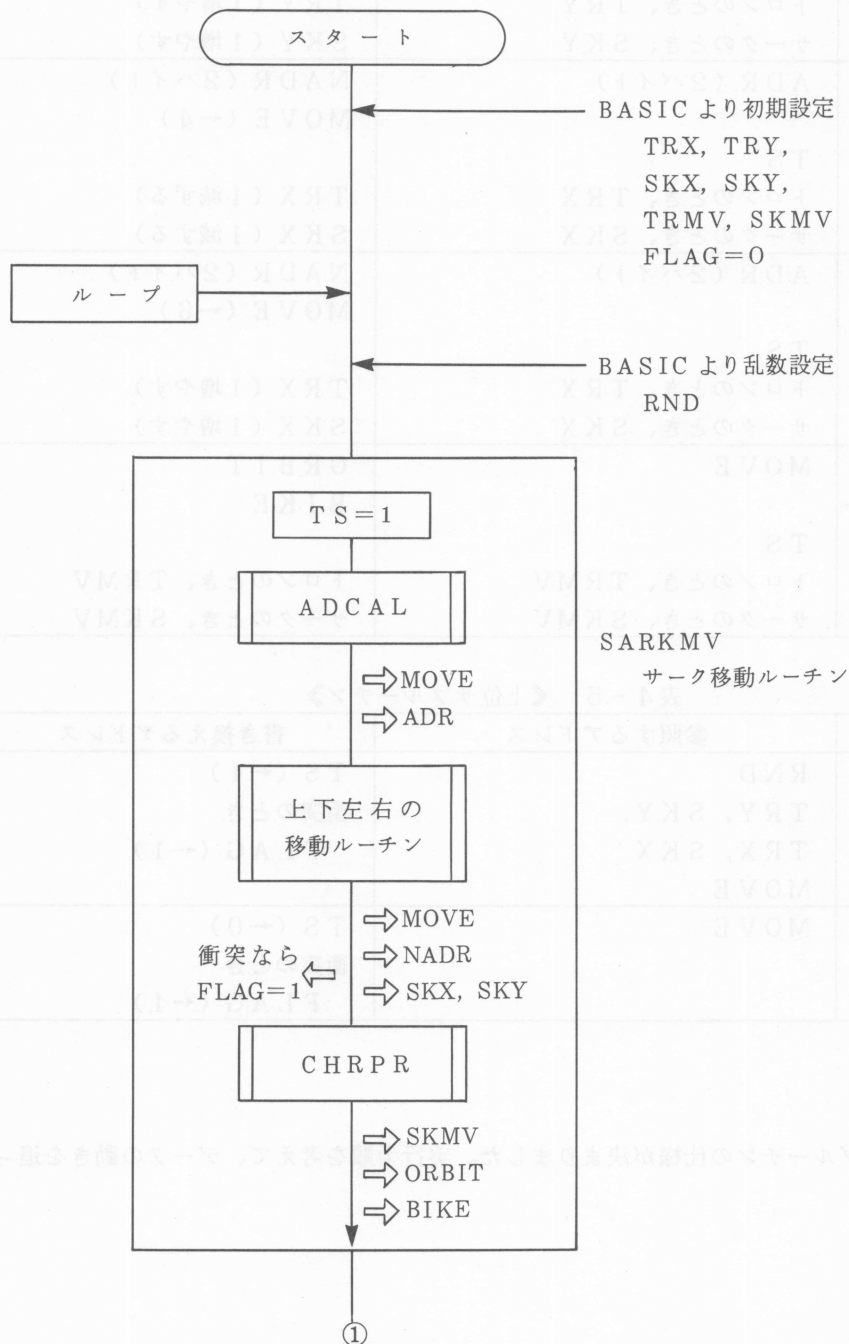
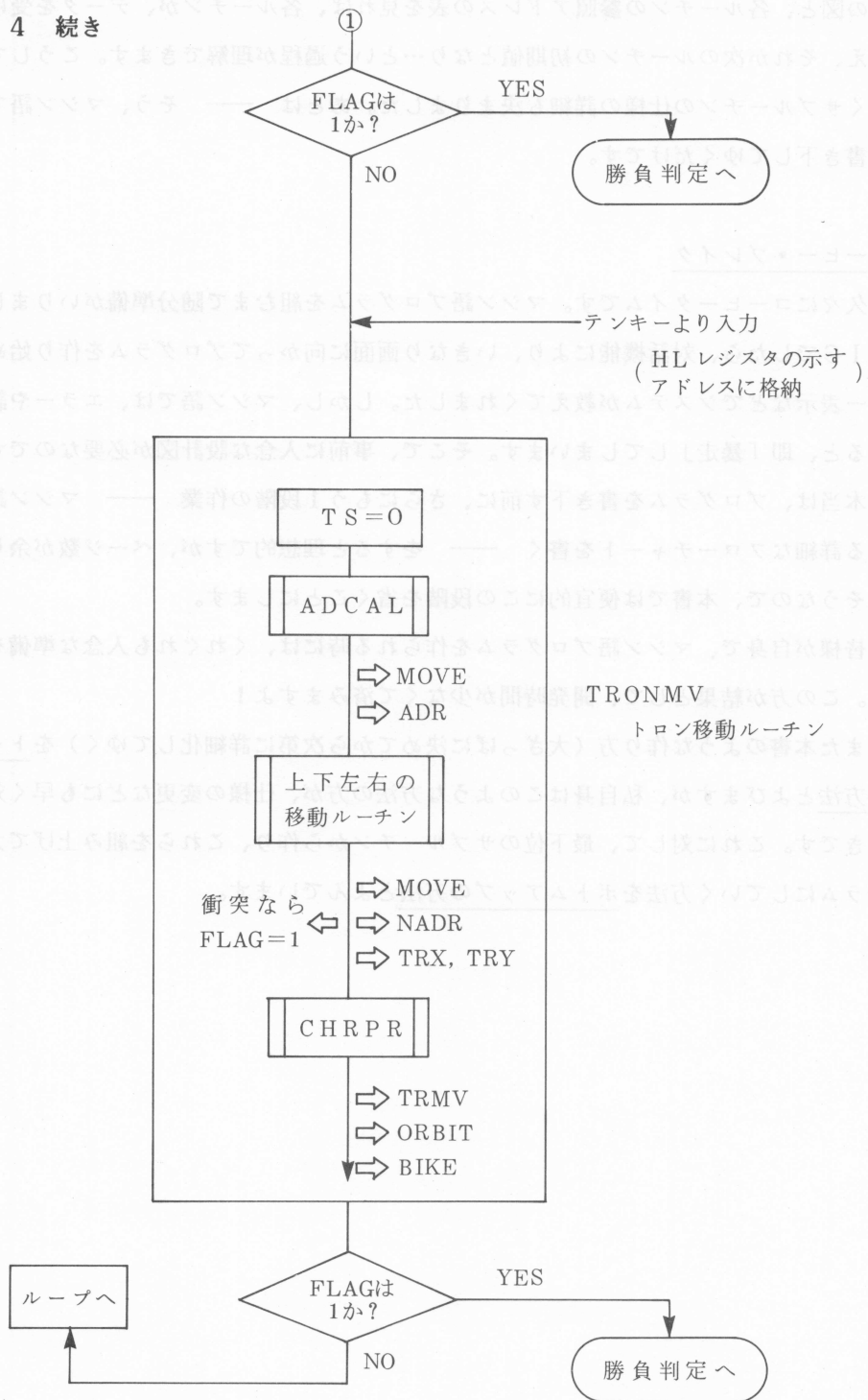
表4-5 《上位サブルーチン》

名　称	参照するアドレス	書き換えるアドレス
SARKMV	RND TRY, SKY, TRX, SKX MOVE	TS(←1) 衝突のとき FLAG(←1)
TRONMV	MOVE	TS(←0) 衝突のとき FLAG(←1)

以上で、サブルーチンの仕様が決まりました。実行手順を考えて、データの動きを追ってみましょう。

図4-14　《データの動き》　　(⇨ は書き換わるアドレス)

図4－14　続き

①
FLAGは1か？
YES
NO
勝負判定へ
テンキーより入力
（HLレジスタの示すアドレスに格納）
TS＝O
ADCAL
MOVE
ADR
TRONMV
トロン移動ルーチン
上下左右の移動ルーチン
MOVE
NADR
TRX, TRY
衝突なら FLAG＝1
CHRPR
TRMV
ORBIT
BIKE
FLAGは1か？
YES
NO
ループへ
勝負判定へ

この図と、各ルーチンの参照アドレスの表を見れば、各ルーチンが、データを受けとり、書き換え、それが次のルーチンの初期値となり…という過程が理解できます。こうして、作成してゆくサブルーチンの仕様の詳細も決まりました。あとは　——　そう、マシン語でプログラムを書き下してゆくだけです。

コーヒー・ブレイク

久々にコーヒータイムです。マシン語プログラムを組むまで随分準備がいりました。ＢＡＳＩＣでしたら、対話機能により、いきなり画面に向かってプログラムを作り始めても、エラー表示などでシステムが教えてくれました。しかし、マシン語では、エラーや設計ミスがあると、即「暴走」してしまいます。そこで、事前に入念な設計図が必要なのです。

本当は、プログラムを書き下す前に、さらにもう１段階の作業　——　マシン語と対応できる詳細なフローチャートを書く　——　をすると理想的ですが、ページ数が余りに多くなりそうなので、本書では便宜的にこの段階を省くことにします。

皆様が自身で、マシン語プログラムを作られる時には、くれぐれも入念な準備をお忘れなく。この方が結果として、開発時間が少なくて済みますよ！

また本書のような作り方（大ざっぱに決めてから次第に詳細化してゆく）をトップダウンの方法とよびますが、私自身はこのような方法の方が、仕様の変更などにも早く対応できて好きです。これに対して、最下位のサブルーチンから作り、これらを組み上げて大きなプログラムにしていく方法をボトムアップの方法とよんでいます。

４－15／サブルーチンSARKMVの作成

まず、サークの移動方向決定サブルーチンである　ＳＡＲＫＭＶ　をマシン語で書き下してみましょう。（紙に書いてもよいし、ＢＡＳＩＣのスクリーンエディット機能を用いて、ＲＥＭ文などで、ＢＡＳＩＣテキストとして書いていってもよいと思います。）

おおよそ、リスト（図４－１５）のようになるはずです。まず、ニーモニックを書いてゆきます。サブルーチンやジャンプ先には適当にラベルをつけて記します。次に、それらを、マシンコードに直し（ハンドアセンブル！）、アドレスを割りふってゆきます。コメントは読者の理解のためにつけましたから、書かなくて構いません。

サブルーチンやジャンプ先のアドレスが未定のうちは、必要なバイト数だけ線を引き、場所をあけておきます。

ルーチンの終わりまで書ききったら、わかる部分について、絶対アドレス、相対アドレスを書き入れてゆきます。たとえば、ＳＥＬＦとラベルをつけたアドレスは、　Ｄ０４４Ｈ　とわかりますね。それから、　Ｄ０６１Ｈ番地の相対ジャンプのジャンプ先は、　Ｄ０６８Ｈ番地になりましたから、　Ｄ０６３Ｈ　を基点０と数えて、５番地先、すなわち相対アドレスは０５Ｈ　となります。もう１つやってみますか。

Ｄ０６６Ｈ番地の相対ジャンプのジャンプ先は、Ｄ０７ＤＨ番地です。　Ｄ０６８Ｈ　を基点０と数えて、２１番地先、すなわちこの相対アドレスは　１５Ｈ　となります。相対アドレスの計算法よろしいですか。残りについても、計算して記入して下さい。

図４－１５

```
■■■ SARKMV ( SARK ﾉ ｲﾄﾞｳ ﾙｰﾁﾝ ) ■■■

ｱﾄﾞﾚｽ ﾏｼﾝｺｰﾄﾞ     ﾆｰﾓﾆｯｸ            ｺﾒﾝﾄ

D000  3E01        LD    A,01H
D002  320DE0      LD    (0E00DH),A  ;TS <-- 1
D005  CD____      CALL  ADCAL       ;ｱﾄﾞﾚｽ ｹｲｻﾝ
D008  3A06E0      LD    A,(0E006H)
D00B  FE04        CP    04H         ;ﾗﾝｽｳ ｦ 4 ﾄ､ ｸﾗﾍﾞﾙ
D00D  DA____      JP    C,SELF      ;ﾗﾝｽｳ < 4 ﾅﾗ､ SARK ﾉ ｼﾞｼｭｹｯﾃｲ ﾍ JUMP
D010  3A01E0      LD    A,(0E001H)  ;TRON ﾉ ｲﾁ ｦ ﾐﾙ ｱﾙｺﾞﾘｽﾞﾑ
D013  2103E0      LD    HL,0E003H
D016  BE          CP    (HL)        ;TRY ﾄ SKY ｦ ｸﾗﾍﾞﾙ
D017  DC____      CALL  C,GOUP      ;TRY < SKY ﾅﾗ､ GOUP ﾍ
D01A  DA____      JP    C,CHRPR     ;GOUP ｼｮﾘ ｦ ｼﾀﾗ､ CHRPR ﾙｰﾁﾝ ﾍ JUMP
D01D  3A01E0      LD    A,(0E001H)
D020  2103E0      LD    HL,0E003H
D023  BE          CP    (HL)        ;TRY ﾄ SKY ｦ ｸﾗﾍﾞﾙ
D024  C4____      CALL  NZ,GODN     ;TRY > SKY ﾅﾗ､ GODN ﾍ
D027  DA____      JP    C,CHRPR     ;GODN ｼｮﾘ ｦ ｼﾀﾗ､ CHRPR ﾙｰﾁﾝ ﾍ JUMP
D02A  3A00E0      LD    A,(0E000H)
D02D  2102E0      LD    HL,0E002H
D030  BE          CP    (HL)        ;TRX ﾄ SKX ｦ ｸﾗﾍﾞﾙ
D031  DC____      CALL  C,GOLT      ;TRX < SKX ﾅﾗ､ GOLT ﾍ
D034  DA____      JP    C,CHRPR     ;GOLT ｼｮﾘ ｦ ｼﾀﾗ､ CHRPR ﾙｰﾁﾝ ﾍ JUMP
D037  3A00E0      LD    A,(0E000H)
D03A  2102E0      LD    HL,0E002H
D03D  BE          CP    (HL)        ;TRX ﾄ SKX ｦ ｸﾗﾍﾞﾙ
D03E  C4____      CALL  NZ,GORT     ;TRX > SKX ﾅﾗ､ GORT ﾍ
D041  DA____      JP    C,CHRPR     ;GORT ｼｮﾘ ｦ ｼﾀﾗ､ CHRPR ﾙｰﾁﾝ ﾍ JUMP
D044  CD____      CALL  GOUP          ;■ SELF ( SARK ﾉ ｼﾞｼｭｹｯﾃｲ ﾙｰﾁﾝ ) ■
D047  DA____      JP    C,CHRPR
D04A  CD____      CALL  GODN
D04D  DA____      JP    C,CHRPR
D050  CD____      CALL  GOLT
D053  DA____      JP    C,CHRPR
D056  CD____      CALL  GORT
D059  DA____      JP    C,CHRPR
                                    ; --> ｲｶ ﾊ､ ｽｽﾒﾙ ﾎｳｺｳ ｶﾞ､ ﾅｲﾄｷ ﾉ ｼｮﾘ

D05C  3A0CE0      LD    A,(0E00CH)  ;■ SARK ｼｮｳﾄﾂ ﾙｰﾁﾝ ■   ** E00CH = MOVE **
D05F  FE08        CP    08H         ;ﾎｳｺｳ ﾊ ｳｴ ｶ ?
D061  20__        JR    NZ, ｼﾀ ﾄ ﾋｶｸ ﾍ  ;ｳｴ ﾃﾞﾅｹﾚﾊﾞ､ ｼﾀ ﾄ ﾋｶｸ ﾍ
D063  CD____      CALL  GOUP1       ;ｳｴ ﾆ ｽｽﾑ
D066  18__        JR    SARK ｼｮｳﾄﾂ ﾍ
D068  FE02        CP    02H         ;ﾎｳｺｳ ﾊ ｼﾀ ｶ ?
D06A  20__        JR    NZ, ﾋﾀﾞﾘ ﾄ ﾋｶｸ ﾍ ;ｼﾀ ﾃﾞﾅｹﾚﾊﾞ､ ﾋﾀﾞﾘ ﾄ ﾋｶｸ ﾍ
D06C  CD____      CALL  GODN1       ;ｼﾀ ﾆ ｽｽﾑ
D06F  18__        JR    SARK ｼｮｳﾄﾂ ﾍ
D071  FE04        CP    04H         ;ﾎｳｺｳ ﾊ ﾋﾀﾞﾘ ｶ ?
D073  20__        JR    NZ, ﾐｷﾞ ﾄ ﾋｶｸ ﾍ ;ﾋﾀﾞﾘ ﾃﾞﾅｹﾚﾊﾞ､ ﾐｷﾞ ﾄ ﾋｶｸ ﾍ
D075  CD____      CALL  GOLT1       ;ﾋﾀﾞﾘ ﾆ ｽｽﾑ
D078  18__        JR    SARK ｼｮｳﾄﾂ ﾍ
D07A  CD____      CALL  GORT1       ;ﾐｷﾞ ﾆ ｽｽﾑ
D07D  3E01        LD    A,01H       ;SARK ｼｮｳﾄﾂ !
D07F  3207E0      LD    (0E007H),A  ;FLAG <- 1
D082  C3____      JP    CHRPR       ;ﾋｮｳｼﾞ ﾙｰﾁﾝ ﾍ JUMP
```

４−16／サブルーチンTRONMVの作成

次に、トロンの移動ルーチン　ＴＲＯＮＭＶ　をマシン語にしましょう。

リスト　（図４－１６）　のようにしてみました。

移動方向を決める部分には、２つの入り口を作りました。１つは、テンキーの入力値をもって、Ｄ０９ＢＨ番地へ入る場合、もう１つはＤ０９８Ｈ番地から入る場合です。後者は、テンキーの値ではなく、現在トロンの進んでいる方向コード（ＴＲＭＶに格納されている）をもって、Ｄ０９ＢＨ番地へ行きます。これは「現状維持」の場合で、テンキー値が、　２，４，６，８　のいずれでもない時は、　ＤＯＤＡＨ番地から、ここへジャンプするようにしました。ですから、　ＤＯＤＡＨ番地の相対ジャンプの相対アドレスは、ＤＯＤＣＨ番地を基点０として、６８番地手前となりますから、－６８を符号つき１６進数で表わして、相対アドレスは　ＢＣＨ　となりますね。

ＳＡＲＫＭＶの時と同様に、この段階でわかる絶対アドレス、相対アドレスを記入して下さい。

図４－１６

```
■■ TRONMV ( TRON ﾉ ｲﾄﾞｳ ﾙｰﾁﾝ ) ■■

ｱﾄﾞﾚｽ ﾏｼﾝｺｰﾄﾞ       ﾆｰﾓﾆｯｸ                ｺﾒﾝﾄ

D085  3E00          LD   A,00H
D087  320DE0        LD   (0E00DH),A       ;TS <── 0
D08A  CD____        CALL ADCAL            ;ｱﾄﾞﾚｽ ｹｲｻﾝ
D08D  3A04E0        LD   A,(0E004H)       ;ｺﾞﾄﾞｳｻ ﾀｲｻｸ  ** E004H = TRMV **
D090  210CE0        LD   HL,0E00CH        ; ** E00CH = MOVE **
D093  86            ADD  A,(HL)           ;A <── A + (HL)
D094  FE0A          CP   0AH              ;A ｦ 10 ﾄ ｸﾗﾍﾞﾙ (TRMV ﾄ MOVE ｶﾞ ﾊﾝﾀｲ ﾎｳｺｳ ﾅﾗ､ 10 ﾆ ﾅﾙ !)
D096  20__          JR   NZ, ｷｰ･ｽｷｬﾝ ﾍ    ;ﾊﾝﾀｲ ﾃﾞﾅｹﾚﾊﾞ､ ｷｰ･ｽｷｬﾝ ﾍ
D098  2104E0        LD   HL,0E004H        ;ｹﾞﾝｼﾞｮｳ ｲｼﾞ

D09B  7E            LD   A,(HL)           ;■■ ｷｰ･ｽｷｬﾝ ■■  ** HL = ﾃﾝｷｰ ﾉ ﾃﾞｰﾀ ﾉ ｱﾙ ｱﾄﾞﾚｽ **
D09C  FE08          CP   08H              ;8 ﾄ ｸﾗﾍﾞﾙ
D09E  20__          JR   NZ, 2 ﾄﾉ ﾋｶｸ ﾍ
D0A0  CD____        CALL GOUP             ;ｳｴ ﾆ ｽｽﾒﾙ ?
D0A3  38__          JR   C,EXIT8          ;GOUP ｼｮﾘ ｦ ｼﾀﾗ､ EXIT8 ﾍ
D0A5  CD____        CALL GOUP1            ;ｳｴ ﾆ ｽｽﾑ ( ｼｮｳﾄﾂ ! )
D0A8  3E01          LD   A,01H
D0AA  3207E0        LD   (0E007H),A       ;FLAG <── 1 ( ｼｮｳﾄﾂ )
D0AD  C3____        JP   CHRPR            ;EXIT8 ( 8 ﾉ ﾙｰﾁﾝ ﾉ ﾃﾞｸﾞﾁ )
D0B0  FE02          CP   02H              ;2 ﾄ ｸﾗﾍﾞﾙ
D0B2  20__          JR   NZ, 4 ﾄﾉ ﾋｶｸ ﾍ
```

図4－16　続き

```
D0B4  CD____      CALL GODN       ;ｼﾀ ﾆ ｽｽﾒﾙ ?
D0B7  38__        JR   C,EXIT2    ;GODN ｼｮﾘ ｦ ｼﾀﾗ、 EXIT2 ﾍ
D0B9  CD____      CALL GODN1      ;ｼﾀ ﾆ ｽｽﾑ ( ｼｮｳﾄﾂ ! )
D0BC  3E01        LD   A,01H
D0BE  3207E0      LD   (0E007H),A ;FLAG <— 1 ( ｼｮｳﾄﾂ )
D0C1  C3____      JP   CHRPR      ;EXIT2
D0C4  FE04        CP   04H        ;4 ﾄ ｸﾗﾍﾞﾙ
D0C6  20__        JR   NZ, 6 ﾄﾉ ﾋｶｸ ﾍ
D0C8  CD____      CALL GOLT       ;ﾋﾀﾞﾘ ﾆ ｽｽﾒﾙ ?
D0CB  38__        JR   C,EXIT4    ;GOLT ｼｮﾘ ｦ ｼﾀﾗ、 EXIT4 ﾍ
D0CD  CD____      CALL GOLT1      ;ﾋﾀﾞﾘ ﾆ ｽｽﾑ ( ｼｮｳﾄﾂ ! )
D0D0  3E01        LD   A,01H
D0D2  3207E0      LD   (0E007H),A ;FLAG <— 1 ( ｼｮｳﾄﾂ )
D0D5  C3____      JP   CHRPR      ;EXIT4
D0D8  FE06        CP   06H        ;6 ﾄ ｸﾗﾍﾞﾙ
D0DA  20__        JR   NZ, ｹﾞﾝｼﾞｮｳ ｲｼﾞ ﾍ;( ｱﾄﾞﾚｽ = D098H )
D0DC  CD____      CALL GORT       ;ﾐｷﾞ ﾆ ｽｽﾒﾙ ?
D0DF  38__        JR   C,EXIT6    ;GORT ｼｮﾘ ｦ ｼﾀﾗ、 EXIT6 ﾍ
D0E1  CD____      CALL GORT1      ;ﾐｷﾞ ﾆ ｽｽﾑ ( ｼｮｳﾄﾂ ! )
D0E4  3E01        LD   A,01H
D0E6  3207E0      LD   (0E007H),A ;FLAG <— 1 ( ｼｮｳﾄﾂ )
D0E9  C3____      JP   CHRPR      ;EXIT6
```

４－17／移動用下位ルーチンの作成

ＴＲＯＮＭＶ　の後には、移動に使われる（すなわち、ＳＡＲＫＭＶ，ＴＲＯＮＭＶから呼び出される）下位サブルーチンを配置することにします。

まず、アドレス計算ルーチン　——　ＡＤＣＡＬ　——　を置きましょう。トロン、サーク共用であることに注意して、書き下すとリスト（図4－17）のようになるでしょう。

新しく出てきた命令がありますから、説明を与えておきます。

解　説

ニーモニック：　ＤＪＮＺ　ｅ　（ｅは符号つき１バイト数値）

マシンコード：　１０　ｅ

機　　　能：　Ｂレジスタの値から１を引き、０にならなければ相対アドレス　ｅ　番地へジャンプ。Ｂレジスタの値が０になったら次へ進む。

ニーモニックの　ＤＪＮＺ　は、　Ｄｅｃｒｅｍｅｎｔ　Ｊｕｍｐ　Ｎｏｎ　Ｚｅｒｏ　の略で、Ｂレジスタをカウンタとして自動的に相対ジャンプしてくれるので、２５５回までのループ処理には便利な命令です。

Ｄ１１４Ｈ番地にこの命令を書きましたが、相対アドレスの計算は、もう慣れましたか？ジャンプ先は　Ｄ１１３Ｈ　ですから、　Ｄ１１６Ｈ　を基点０として、３番地手前となり、－３　を符号つき１６進表示して　ＦＤＨ　が相対アドレスですね。

ところで、このループは何をしているかおわかりですか？　ＨＬレジスタペアの初期値は　３０００Ｈ　です。また、ＤＥレジスタペアには、（トロンまたはサークの）ｙ座標が入っています。　ＨＬ　←　ＨＬ＋ＤＥ　を４０回くり返すのですから、結局、ＨＬレジスタペアには　３０００Ｈ＋ｙ＊４０　という計算の答が入ります。

これに、　ＡＤＤ　ＨＬ，ＢＣ　を行なうと、Ｂレジスタは今は０であることに注意し、ＢＣにはｘ座標が入っている訳ですから、ＨＬレジスタペアには　３０００Ｈ＋ｙ＊４０＋ｘ　すなわち、ＶＲＡＭのアドレスが入ることになるのですね。

このように、「かけ算」の１つの方法は、たし算のくり返しで実現できることを覚えておきましょう。

また説明を特にしませんでしたが、１６ビットの加算命令

```
ADD  HL, DE
ADD  HL, BC
```

についてはよいですね。Ａレジスタが、８ビットのアキュームレータであったのと同様に、ＨＬレジスタペアは、１６ビットのアキュームレータとしての役割も持っていることに注意しましょう。ですから、１６ビットの計算では普通、「ＨＬレジスタペアに対して加減を行ない、結果をＨＬに格納する」という形をとることを覚えておいて下さい。

また、Ｄ１１７Ｈ番地の　ＬＤ　（０Ｅ００８Ｈ），ＨＬ　もよろしいですか？　Ｚ８０の鉄則で、上下位逆転が起こって、

Ｅ００８Ｈ番地　←　Ｌレジスタの値が入る

（ＶＲＡＭのアドレス下位バイト）

Ｅ００９Ｈ番地　←　Ｈレジスタの値が入る

（ＶＲＡＭのアドレス上位バイト）

となりますね。

相対アドレスをすべて記入して下さい。

図４－１７

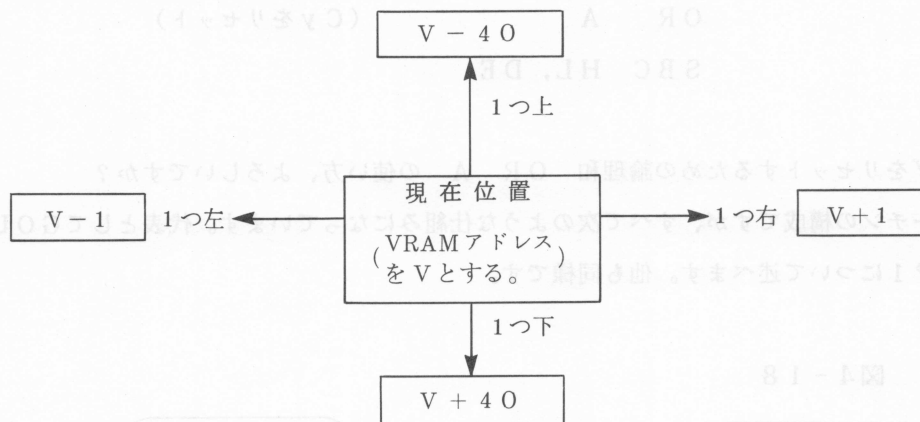
```
■■ ADCAL ( ｱﾄﾞﾚｽ ｹｲｻﾝ ｻﾌﾞﾙｰﾁﾝ ) ■■

ｱﾄﾞﾚｽ ﾏｼﾝｺｰﾄﾞ   ﾆｰﾓﾆｯｸ                ｺﾒﾝﾄ

D0EC  3A0DE0    LD    A,(0E00DH)      ; ** E00DH = TS
D0EF  FE00      CP    00H             ;ﾄﾛﾝ ｶ ?
D0F1  20__      JR    NZ, ｻｰｸ ﾉ ｼｮﾘ ﾍ
D0F3  7E        LD    A,(HL)          ;ﾄﾛﾝ ﾉ ｼｮﾘ  ** HL = ﾃﾝｷｰ ﾉ ｱﾀｲ ﾉ ｱﾙ ｱﾄﾞﾚｽ **
D0F4  320CE0    LD    (0E00CH),A      ; ** E00CH = MOVE **
D0F7  2100E0    LD    HL,0E000H       ; ** E000H = TRX **
D0FA  56        LD    D,(HL)          ;D <── TRX ﾉ ｱﾀｲ
D0FB  23        INC   HL              ;HL <── HL + 1
D0FC  5E        LD    E,(HL)          ;E <── TRY ﾉ ｱﾀｲ
D0FD  18__      JR    ｱﾄﾞﾚｽ ｹｲｻﾝ ﾍ
D0FF  3A05E0    LD    A,(0E005H)      ;ｻｰｸ ﾉ ｼｮﾘ  ** E005H = SKMV **
D102  320CE0    LD    (0E00CH),A      ; ** E00CH = MOVE **
D105  2102E0    LD    HL,0E002H       ; ** E002H = SKX **
D108  56        LD    D,(HL)          ;D <── SKX ﾉ ｱﾀｲ
D109  23        INC   HL              ;HL <── HL + 1
D10A  5E        LD    E,(HL)          ;E <── SKY ﾉ ｱﾀｲ

D10B  210030    LD    HL,3000H        ;ｱﾄﾞﾚｽ ｹｲｻﾝ ** 3000H = VRAM ﾄｯﾌﾟ ｱﾄﾞﾚｽ **
D10E  4A        LD    C,D             ;D ﾚｼﾞｽﾀ ｦ C ﾚｼﾞｽﾀ ﾍ ﾀｲﾋ ｻｾﾙ
D10F  1600      LD    D,00H           ;DE = Y ｻﾞﾋｮｳ ﾉ ｱﾀｲ
D111  0628      LD    B,28H           ;B = ﾙｰﾌﾟ ｶｲｽｳ 40 ｶｲ
D113  19        ADD   HL,DE           ;HL <── HL + DE
D114  10__      DJNZ  D113H ﾍ         ;B ﾚｼﾞｽﾀ ｦ ｶｳﾝﾀｰ ﾄｼﾃ ﾙｰﾌﾟ ｻｾﾙ
D116  09        ADD   HL,BC           ;HL <── 3000H + Y*40 + X  ** VRAM ﾉ ｱﾄﾞﾚｽ **
D117  2208E0    LD    (0E008H),HL     ; ** E008H = ADR **
D11A  C9        RET
```

続いて、ＧＯＵＰ，ＧＯＵＰ１，ＧＯＤＮ，ＧＯＤＮ１，ＧＯＬＴ，ＧＯＬＴ１，ＧＯＲＴ，ＧＯＲＴ１　の上下左右移動サブルーチンを書くことにします。

移動とＶＲＡＭアドレスについて注意しておきます。

従って、ＨＬレジスタペアに現在位置のＶＲＡＭアドレスの値を入れるとき、次により各アドレスを計算できます。

１つ下の位置	：	ＬＤ　ＤＥ，００２８Ｈ　（１０進で４０）
		ＡＤＤ　ＨＬ，ＤＥ
１つ左の位置	：	ＤＥＣ　ＨＬ
１つ右の位置	：	ＩＮＣ　ＨＬ

注意すべきなのは、「１つ上の位置」を計算するものです。　ＨＬ←ＨＬ―ＤＥ　にあたる命令があるとよいのですが、「Ｚ８０命令表」にはありません。近いものとしては次があります。

解　説

ニーモニック：　ＳＢＣ　ＨＬ，ＤＥ

マシンコード：　ＥＤ　５２

機　　　能：　ＨＬ←ＨＬ―ＤＥ―Ｃｙ

すなわち、Ｃｙフラグ込みでの１６ビット減算命令です。従って、Ｃｙフラグを０にした後

に使用すれば目的を達することができます。

1つ上の位置　：　ＬＤ　　ＤＥ，００２８Ｈ　（１０進で４０）
　　　　　　　　　ＯＲ　　Ａ　　　　　　　（Ｃｙをリセット）
　　　　　　　　　ＳＢＣ　ＨＬ，ＤＥ

Ｃｙフラグをリセットするための論理和　ＯＲ　Ａ　の使い方、よろしいですか？

各移動ルーチンの構成ですが、すべて次のような仕組みになっています。代表としてＧＯＵＰ，ＧＯＵＰ１について述べます。他も同様です。

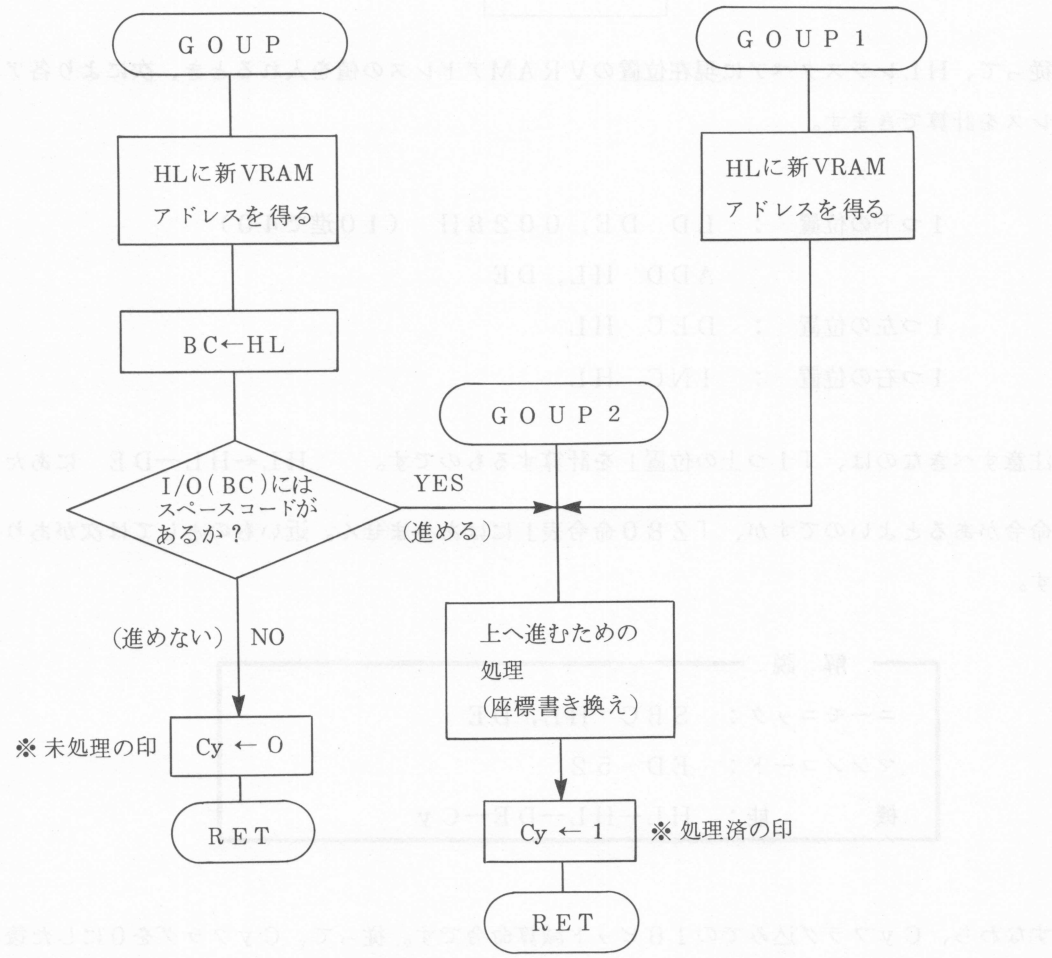
図４－１８

次に、各ルーチンのリストを掲げます。相対アドレスを記入して下さい。

図4－19～22

■ GOUP （ ﾊﾝﾀﾞﾝ ﾂｷ ﾃﾞ ｳｴ ﾆ ｽｽﾑ ｻﾌﾞﾙｰﾁﾝ ） ■　（図4－19①）

```
ｱﾄﾞﾚｽ ﾏｼﾝｺｰﾄﾞ     ﾆｰﾓﾆｯｸ               ｺﾒﾝﾄ

D11B  2A08E0      LD   HL,(0E008H)     ; ** E008H = ADR **
D11E  112800      LD   DE,0028H        ;DE <-- 40
D121  B7          OR   A               ;Cy <-- 0
D122  ED52        SBC  HL,DE           ;HL <-- HL - 40
D124  44          LD   B,H
D125  4D          LD   C,L             ;BC <-- HL
D126  ED78        IN   A,(C)           ;A <-- I/O(BC)
D128  FE20        CP   20H             ;ｽﾍﾟｰｽ･ｺｰﾄﾞ ｶ ?
D12A  28__        JR   Z, GOUP2        ;ｽｽﾒﾙ ﾅﾗ､ GOUP2 ﾍ
D12C  B7          OR   A               ;Cy <-- 0  ･･･ ｽｽﾒﾅｲﾅﾗ､ Cyﾌﾗｸﾞ ｦ ﾘｾｯﾄ ｽﾙ
D12D  C9          RET
```

■ GOUP1 （ ﾑｼﾞｮｳｹﾝ ﾆ ｳｴ ﾆ ｽｽﾑ ｻﾌﾞﾙｰﾁﾝ ） ■　（図4－19②）

```
ｱﾄﾞﾚｽ ﾏｼﾝｺｰﾄﾞ     ﾆｰﾓﾆｯｸ               ｺﾒﾝﾄ

D12E  2A08E0      LD   HL,(0E008H)
D131  112800      LD   DE,0028H
D134  B7          OR   A
D135  ED52        SBC  HL,DE           ;ｺｺﾏﾃﾞﾊ､ GOUP ﾄ ｵﾅｼﾞ

D137  220AE0      LD   (0E00AH),HL     ;GOUP2  ** E00AH = NADR **
D13A  3E08        LD   A,08H
D13C  320CE0      LD   (0E00CH),A      ;MOVE ﾆ ｢ｳｴ｣ ﾉ ｺｰﾄﾞ 8 ｦ ｲﾚﾙ
D13F  3A0DE0      LD   A,(0E00DH)      ; ** E00DH = TS **
D142  FE00        CP   00H             ;ﾄﾛﾝ ｶ ?
D144  20__        JR   NZ, ｻｰｸ･ｳｴ ﾍ
D146  3A01E0      LD   A,(0E001H)      ;ﾄﾛﾝ･ｳｴ ﾉ ｼｮﾘ  ** E001H = TRY **
D149  3D          DEC  A
D14A  3201E0      LD   (0E001H),A      ;TRY ﾉ ｱﾀｲ <-- TRY ﾉ ｱﾀｲ - 1
D14D  18__        JR   ｳｴ･ｼｮﾘ ｵﾜﾘ ﾍ
D14F  3A03E0      LD   A,(0E003H)      ;ｻｰｸ･ｳｴ ﾉ ｼｮﾘ  ** E003H = SKY **
D152  3D          DEC  A
D153  3203E0      LD   (0E003H),A      ;SKY ﾉ ｱﾀｲ <-- SKY ﾉ ｱﾀｲ - 1
D156  37          SCF                  ;ｳｴ･ｼｮﾘ ｵﾜﾘ  ･･ Cyﾌﾗｸﾞ ｦ ｾｯﾄ ｽﾙ
D157  C9          RET
```

■■ GODN (ﾊﾝﾀﾞﾝ ﾂｷ ﾃﾞ ｼﾀ ﾆ ｽｽﾑ ｻﾌﾞﾙｰﾁﾝ) ■■　**（図４－２０①）**

```
ｱﾄﾞﾚｽ ﾏｼﾝｺｰﾄﾞ      ﾆｰﾓﾆｯｸ                    ｺﾒﾝﾄ

D158  2A08E0       LD    HL,0E008H          ; ** E008H = ADR **
D15B  112800       LD    DE,0028H           ;DE <-- 40
D15E  19           ADD   HL,DE              ;HL <-- HL + 40
D15F  44           LD    B,H
D160  4D           LD    C,L                ;BC <-- HL
D161  ED78         IN    A,(C)              ;A <-- I/O(BC)
D163  FE20         CP    20H                ;ｽﾍﾟｰｽ･ｺｰﾄﾞ ｶ ?
D165  28__         JR    Z, GODN2           ;ｽｽﾒﾙ ﾅﾗ､ GODN2 ﾍ
D167  B7           OR    A                  ;Cy <-- 0  ･･･ ｽｽﾒﾅｲﾅﾗ､ Cyﾌﾗｸﾞ ｦ ﾘｾｯﾄ ｽﾙ
D168  C9           RET
```

■■ GODN1 (ﾑｼﾞｮｳｹﾝ ﾆ ｼﾀ ﾆ ｽｽﾑ ｻﾌﾞﾙｰﾁﾝ) ■■　**（図４－２０②）**

```
ｱﾄﾞﾚｽ ﾏｼﾝｺｰﾄﾞ      ﾆｰﾓﾆｯｸ                    ｺﾒﾝﾄ

D169  2A08E0       LD    HL,(0E008H)
D16C  112800       LD    DE,0028H
D16F  19           ADD   HL,DE              ;ｺｺﾏﾃﾞﾊ､ GODN ﾄ ｵﾅｼﾞ

D170  220AE0       LD    (0E00AH),HL        ;GODN2  ** E00AH = NADR **
D173  3E02         LD    A,02H
D175  320CE0       LD    (0E00CH),A         ;MOVE ﾆ ｢ｼﾀ｣ ﾉ ｺｰﾄﾞ 2 ｦ ｲﾚﾙ
D178  3A0DE0       LD    A,(0E00DH)         ; ** E00DH = TS **
D17B  FE00         CP    00H                ;ﾄﾛﾝ ｶ ?
D17D  20__         JR    NZ, ｻｰｸ･ｼﾀ ﾍ
D17F  3A01E0       LD    A,(0E001H)         ;ﾄﾛﾝ･ｼﾀ ﾉ ｼｮﾘ  ** E001H = TRY **
D182  3C           INC   A
D183  3201E0       LD    (0E001H),A         ;TRY ﾉ ｱﾀｲ <-- TRY ﾉ ｱﾀｲ + 1
D186  18__         JR    ｼﾀ･ｼｮﾘ ｵﾜﾘ ﾍ
D188  3A03E0       LD    A,(0E003H)         ;ｻｰｸ･ｼﾀ ﾉ ｼｮﾘ  ** E003H = SKY **
D18B  3C           INC   A
D18C  3203E0       LD    (0E003H),A         ;SKY ﾉ ｱﾀｲ <-- SKY ﾉ ｱﾀｲ + 1
D18F  37           SCF                      ;ｼﾀ･ｼｮﾘ ｵﾜﾘ ･･ Cyﾌﾗｸﾞ ｦ ｾｯﾄ ｽﾙ
D190  C9           RET
```

■■ GOLT (ﾊﾝﾀﾞﾝ ﾂｷ ﾃﾞ ﾋﾀﾞﾘ ﾆ ｽｽﾑ ｻﾌﾞﾙｰﾁﾝ) ■■　**（図４－２１①）**

```
ｱﾄﾞﾚｽ ﾏｼﾝｺｰﾄﾞ      ﾆｰﾓﾆｯｸ                    ｺﾒﾝﾄ

D191  2A08E0       LD    HL,(0E008H)        ; ** E008H = ADR **
D194  2B           DEC   HL                 ;HL <-- HL - 1
D195  44           LD    B,H
D196  4D           LD    C,L                ;BC <-- HL
D197  ED78         IN    A,(C)              ;A <-- I/O(BC)
D199  FE20         CP    20H                ;ｽﾍﾟｰｽ･ｺｰﾄﾞ ｶ ?
D19B  28__         JR    Z, GOLT2           ;ｽｽﾒﾙ ﾅﾗ､ GOLT2 ﾍ
D19D  B7           OR    A                  ;Cy <-- 0  ･･･ ｽｽﾒﾅｲﾅﾗ､ Cyﾌﾗｸﾞ ｦ ﾘｾｯﾄ ｽﾙ
D19E  C9           RET
```

■ GOLT1 (ﾑｼﾞｮｳｹﾝ ﾆ ﾋﾀﾞﾘ ﾆ ｽｽﾑ ｻﾌﾞﾙｰﾁﾝ) ■ （図4－21②）

```
ｱﾄﾞﾚｽ ﾏｼﾝｺｰﾄﾞ      ﾆｰﾓﾆｯｸ                ｺﾒﾝﾄ

D19F  2A08E0      LD    HL,(0E008H)
D1A2  2B          DEC   HL               ;ｺｺﾏﾃﾞﾊ、GOLT ﾄ ｵﾅｼﾞ

D1A3  220AE0      LD    (0E00AH),HL      ;GOLT2  ** E00AH = NADR **
D1A6  3E04        LD    A,04H
D1A8  320CE0      LD    (0E00CH),A       ;MOVE ﾆ「ﾋﾀﾞﾘ」ﾉ ｺｰﾄﾞ 4 ｦ ｲﾚﾙ
D1AB  3A0DE0      LD    A,(0E00DH)       ; ** E00DH = TS **
D1AE  FE00        CP    00H              ;ﾄﾛﾝ ｶ ?
D1B0  20__        JR    NZ, ｻｰｸ･ﾋﾀﾞﾘ ﾍ
D1B2  3A00E0      LD    A,(0E000H)       ;ﾄﾛﾝ･ﾋﾀﾞﾘ ﾉ ｼｮﾘ  ** E000H = TRX **
D1B5  3D          DEC   A
D1B6  3200E0      LD    (0E000H),A       ;TRX ﾉ ｱﾀｲ <— TRX ﾉ ｱﾀｲ - 1
D1B9  18__        JR    ﾋﾀﾞﾘ･ｼｮﾘ ｵﾜﾘ ﾍ
D1BB  3A02E0      LD    A,(0E002H)       ;ｻｰｸ･ﾋﾀﾞﾘ ﾉ ｼｮﾘ  ** E002H = SKX **
D1BE  3D          DEC   A
D1BF  3202E0      LD    (0E002H),A       ;SKX ﾉ ｱﾀｲ <— SKX ﾉ ｱﾀｲ - 1
D1C2  37          SCF                    ;ﾋﾀﾞﾘ･ｼｮﾘ ｵﾜﾘ  ･･ Cyﾌﾗｸﾞ ｦ ｾｯﾄ ｽﾙ
D1C3  C9          RET
```

■ GORT (ﾊﾝﾀﾞﾝ ﾂｷ ﾃﾞ ﾐｷﾞ ﾆ ｽｽﾑ ｻﾌﾞﾙｰﾁﾝ) ■ （図4－22①）

```
ｱﾄﾞﾚｽ ﾏｼﾝｺｰﾄﾞ      ﾆｰﾓﾆｯｸ                ｺﾒﾝﾄ

D1C4  2A08E0      LD    HL,(0E008H)      ; ** E008H = ADR **
D1C7  23          INC   HL               ;HL <— HL + 1
D1C8  44          LD    B,H
D1C9  4D          LD    C,L              ;BC <— HL
D1CA  ED78        IN    A,(C)            ;A <— I/O(BC)
D1CC  FE20        CP    20H              ;ｽﾍﾟｰｽ･ｺｰﾄﾞ ｶ ?
D1CE  28__        JR    Z, GORT2         ;ｽｽﾒﾙ ﾅﾗ、GORT2 ﾍ
D1D0  B7          OR    A                ;Cy <— 0   ･･･ ｽｽﾒﾅｲﾅﾗ、Cyﾌﾗｸﾞ ｦ ﾘｾｯﾄ ｽﾙ
D1D1  C9          RET
```

■ GORT1 (ﾑｼﾞｮｳｹﾝ ﾆ ﾐｷﾞ ﾆ ｽｽﾑ ｻﾌﾞﾙｰﾁﾝ) ■ （図4－22②）

```
ｱﾄﾞﾚｽ ﾏｼﾝｺｰﾄﾞ      ﾆｰﾓﾆｯｸ                ｺﾒﾝﾄ

D1D2  2A08E0      LD    HL,(0E008H)
D1D5  23          INC   HL               ;ｺｺﾏﾃﾞﾊ、GORT ﾄ ｵﾅｼﾞ

D1D6  220AE0      LD    (0E00AH),HL      ;GORT2  ** E00AH = NADR **
D1D9  3E06        LD    A,06H
D1DB  320CE0      LD    (0E00CH),A       ;MOVE ﾆ「ﾐｷﾞ」ﾉ ｺｰﾄﾞ 6 ｦ ｲﾚﾙ
D1DE  3A0DE0      LD    A,(0E00DH)       ; ** E00DH = TS **
D1E1  FE00        CP    00H              ;ﾄﾛﾝ ｶ ?
D1E3  20__        JR    NZ, ｻｰｸ･ﾐｷﾞ ﾍ
D1E5  3A00E0      LD    A,(0E000H)       ;ﾄﾛﾝ･ﾐｷﾞ ﾉ ｼｮﾘ  ** E000H = TRX **
D1E8  3C          INC   A
D1E9  3200E0      LD    (0E000H),A       ;TRX ﾉ ｱﾀｲ <— TRX ﾉ ｱﾀｲ + 1
D1EC  18__        JR    ﾐｷﾞ･ｼｮﾘ ｵﾜﾘ ﾍ
D1EE  3A02E0      LD    A,(0E002H)       ;ｻｰｸ･ﾐｷﾞ ﾉ ｼｮﾘ  ** E002H = SKX **
D1F1  3C          INC   A
D1F2  3202E0      LD    (0E002H),A       ;SKX ﾉ ｱﾀｲ <— SKX ﾉ ｱﾀｲ + 1
D1F5  37          SCF                    ;ﾐｷﾞ･ｼｮﾘ ｵﾜﾘ  ･･ Cyﾌﾗｸﾞ ｦ ｾｯﾄ ｽﾙ
D1F6  C9          RET
```

４−18／表示ルーチンCHRPRの作成

続けて、マシン語ルーチン後半部の作成にかかりましょう。まず、移動ルーチンからのジャンプ先に指定しておいた表示ルーチン　ＣＨＲＰＲ　を配置いたします。

４－１２節で設計した方針で、ＣＨＲＰＲ　ルーチンは、リスト（図４－２３）のように書き下してみました。

１つ注意すべき点を述べます。それは、プリント・ルーチン（ＰＲＩＮＴ　と名づける）へのデータの引き渡し方です。ここでは、４－１４節で触れたように、レジスタを介してデータを渡す方法を採用してみました。すなわち、ＰＲＩＮＴサブルーチンは次の仕様といたします。

《ＰＲＩＮＴサブルーチンの仕様》

入力条件　：　Ｅレジスタ　＝　表示したいキャラクターのＡＳＣＩＩコード
Ｄレジスタ　＝　キャラクター識別コード（００Ｈ＝軌跡キャラクター，
０１Ｈ＝バイクキャラクター，０２Ｈ＝衝突キャラクター）
ＢＣレジスタペア　＝　表示位置のＶＲＡＭアドレス

機　能　：　ＤレジスタとＴＳフラグ（トロン・サーク識別フラグ）を参照して適切なアトリビュートを決定し、Ｅレジスタのコードを、ＢＣレジスタペアのＶＲＡＭアドレスに出力する。

このことを念頭に置いて、リストを眺めると、何をしているかがわかると思います。
また、ＰＲＩＮＴルーチンへジャンプする所がありますが、それは、サブルーチン内では、

```
JP PRINT = CALL PRINT
           RET
```

という同値性が成立することによっています。

図4－23

```
■ CHRPR ( ｷｬﾗｸﾀｰ ﾋｮｳｼﾞ ｻﾌﾞﾙｰﾁﾝ ) ■

ｱﾄﾞﾚｽ ﾏｼﾝｺｰﾄﾞ       ﾆｰﾓﾆｯｸ                ｺﾒﾝﾄ

D1F7  CD____     CALL CHRDET           ;ｷｬﾗｸﾀｰ ﾉ ｾﾝﾃｲ ﾙｰﾁﾝ ｦ ﾖﾋﾞﾀﾞｽ
D1FA  ED4B08E0   LD   BC,(0E008H)      ; ** E008H = ADR **
D1FE  1600       LD   D,00H            ;ｷｾｷ ﾉ ｷｬﾗｸﾀｰ ｦ ｼﾃｲ
D200  3A0EE0     LD   A,(0E00EH)       ; ** E00EH = ORBIT **
D203  5F         LD   E,A              ;E <-- ｷｾｷ ﾉ ASCII ｺｰﾄﾞ
D204  CD____     CALL PRINT            ;ﾌﾟﾘﾝﾄ ﾙｰﾁﾝ ｦ ﾖﾋﾞﾀﾞｽ
D207  3A07E0     LD   A,(0E007H)       ; ** E007H = FLAG **
D20A  FE00       CP   00H              ;ｼｮｳﾄﾂ ｶ ?
D20C  20__       JR   NZ, ｼｮｳﾄﾂ･ｷｬﾗｸﾀｰ ｼｮﾘ ﾍ
D20E  ED4B0AE0   LD   BC,(0E00AH)      ;ﾊﾞｲｸ･ｷｬﾗｸﾀｰ ｼｮﾘ  ** E00AH = NADR **
D212  1601       LD   D,01H            ;ﾊﾞｲｸ ﾉ ｷｬﾗｸﾀｰ ｦ ｼﾃｲ
D214  3A0FE0     LD   A,(0E00FH)       ; ** E00FH = BIKE **
D217  5F         LD   E,A              ;E <-- ﾊﾞｲｸ ﾉ ASCIIｺｰﾄﾞ
D218  C3____     JP   PRINT            ;ﾌﾟﾘﾝﾄ ﾙｰﾁﾝ ﾍ
D21B  ED4B0AE0   LD   BC,(0E00AH)      ;ｼｮｳﾄﾂ･ｷｬﾗｸﾀｰ ｼｮﾘ  ** E00AH = NADR **
D21F  1602       LD   D,02H            ;ｼｮｳﾄﾂ ﾉ ｷｬﾗｸﾀｰ ｦ ｼﾃｲ
D221  1EE8       LD   E,0E8H           ;E <-- X ﾉ ASCII ｺｰﾄﾞ(E8H)
D223  C3____     JP   PRINT            ;ﾌﾟﾘﾝﾄ ﾙｰﾁﾝ ﾍ
```

４−19／キャラクター選定サブルーチンCHRDETの作成

次に　ＣＨＲＰＲ　から呼び出される１つ目の下位サブルーチン　ＣＨＲＤＥＴ　を書き下しましょう。４－１２節で設計した仕様に従って作成します。

続いて、下位サブルーチン　ＣＨＲＵＰ，ＣＨＲＤＮ，ＣＨＲＬＴ，ＣＨＲＲＴ　を配置いたします。

リスト　（図４－２４から図４－２７）　のようになります。

図４－２４

```
■■ CHRDET ( ｷｬﾗｸﾀｰ ｹｯﾃｲ ﾙｰﾁﾝ ) ■■

ｱﾄﾞﾚｽ ﾏｼﾝｺｰﾄﾞ     ﾆｰﾓﾆｯｸ                    ｺﾒﾝﾄ

D226  3A0CE0      LD    A,(0E00CH)          ; ** E00CH = MOVE **
D229  FE02        CP    02H                 ;ｲﾄﾞｳ ﾎｳｺｳ ｦ ｼﾗﾍﾞﾙ
D22B  CA____      JP    Z, ｼﾀ・ｷｬﾗｸﾀｰ ﾍ      ;ﾎｳｺｳ ｶﾞ 2 ﾉ ﾄｷ
D22E  FE04        CP    04H
D230  CA____      JP    Z, ﾋﾀﾞﾘ・ｷｬﾗｸﾀｰ ﾍ ;ﾎｳｺｳ ｶﾞ 4 ﾉ ﾄｷ
D233  FE06        CP    06H
D235  CA____      JP    Z, ﾐｷﾞ・ｷｬﾗｸﾀｰ ﾍ    ;ﾎｳｺｳ ｶﾞ 6 ﾉ ﾄｷ

D238  3A0DE0      LD    A,(0E00DH)          ;ｳｴ・ｷｬﾗｸﾀｰ (CHRUP)  ** E00DH = TS **
D23B  FE00        CP    00H                 ;ﾄﾛﾝ ｶ ?
D23D  20__        JR    NZ, ｻｰｸ・ｳｴ・ｷｾｷ ﾍ
D23F  3A04E0      LD    A,(0E004H)          ;ﾄﾛﾝ・ｳｴ・ｷｾｷ ﾉ ｼｮﾘ  ** E004H = TRMV **
D242  18__        JR    ｢ﾋﾀﾞﾘ ｶﾗ ｳｴ ?｣ ﾍ
D244  3A05E0      LD    A,(0E005H)          ;ｻｰｸ・ｳｴ・ｷｾｷ ﾉ ｼｮﾘ  ** E005H = SKMV **

D247  FE04        CP    04H                 ;｢ﾋﾀﾞﾘ ｶﾗ ｳｴ ?｣
D249  20__        JR    NZ, ｢ﾐｷﾞ ｶﾗ ｳｴ ?｣ ﾍ
D24B  3E99        LD    A,99H               ;A <— └ ﾉ ｺｰﾄﾞ
D24D  18__        JR    ｳｴ・ｷｾｷ ﾍ
D24F  FE06        CP    06H                 ;｢ﾐｷﾞ ｶﾗ ｳｴ ?｣
D251  20__        JR    NZ, ｢ｳｴ ﾉ ﾏﾏ｣ ﾍ
D253  3E98        LD    A,98H               ;A <— ┘ ﾉ ｺｰﾄﾞ
D255  18__        JR    ｳｴ・ｷｾｷ ﾍ
D257  3E91        LD    A,91H               ;｢ｳｴ ﾉ ﾏﾏ｣ ‥ A <— │ ﾉ ｺｰﾄﾞ
D259  320EE0      LD    (0E00EH),A          ;ｳｴ・ｷｾｷ ﾉ ｼｮﾘ  ** E00EH = ORBIT **
D25C  3A0DE0      LD    A,(0E00DH)          ; ** E00DH = TS **
D25F  FE00        CP    00H                 ;ﾄﾛﾝ ｶ ?
D261  20__        JR    NZ, ｻｰｸ・ｳｴ・ﾊﾞｲｸ ﾍ

D263  3E64        LD    A,64H               ;ﾄﾛﾝ・ｳｴ・ﾊﾞｲｸ ‥ 64H = 100
D265  320FE0      LD    (0E00FH),A          ;BIKE ﾆ 100 ｦ ｶｸﾉｳ ｽﾙ
D268  3E08        LD    A,08H               ; ** 8 = ｳｴ ﾉ ｺｰﾄﾞ **
D26A  3204E0      LD    (0E004H),A          ;TRMV ﾆ 8 ｦ ｶｸﾉｳ ｽﾙ
D26D  C9          RET

D26E  3EC8        LD    A,0C8H              ;ｻｰｸ・ｳｴ・ﾊﾞｲｸ ‥ C8H = 200
D270  320FE0      LD    (0E00FH),A          ;BIKE ﾆ 200 ｦ ｶｸﾉｳ ｽﾙ
D273  3E08        LD    A,08H               ; ** 8 = ｳｴ ﾉ ｺｰﾄﾞ **
D275  3205E0      LD    (0E005H),A          ;SKMV ﾆ 8 ｦ ｶｸﾉｳ ｽﾙ
D278  C9          RET
```

```
ｱﾄﾞﾚｽ ﾏｼﾝｺｰﾄﾞ      ﾆｰﾓﾆｯｸ                 ｺﾒﾝﾄ

D279  3A0DE0      LD   A,(0E00DH)          ;ｼﾀ･ｷｬﾗｸﾀｰ (CHRDN)  ※※ E00DH = TS ※※
D27C  FE00        CP   00H                 ;ﾄﾛﾝ ｶ ?
D27E  20__        JR   NZ, ｻｰｸ･ｼﾀ･ｷｾｷ ﾍ
D280  3A04E0      LD   A,(0E004H)          ;ﾄﾛﾝ･ｼﾀ･ｷｾｷ ﾉ ｼｮﾘ  ※※ E004H = TRMV ※※
D283  18__        JR   ｢ﾋﾀﾞﾘ ｶﾗ ｼﾀ ?｣ ﾍ
D285  3A05E0      LD   A,(0E005H)          ;ｻｰｸ･ｼﾀ･ｷｾｷ ﾉ ｼｮﾘ  ※※ E005H = SKMV ※※

D288  FE04        CP   04H                 ;｢ﾋﾀﾞﾘ ｶﾗ ｼﾀ ?｣
D28A  20__        JR   NZ, ｢ﾐｷﾞ ｶﾗ ｼﾀ ?｣ ﾍ
D28C  3E9A        LD   A,9AH               ;A <── ┌ ﾉ ｺｰﾄﾞ
D28E  18__        JR   ｼﾀ･ｷｾｷ ﾍ
D290  FE06        CP   06H                 ;｢ﾐｷﾞ ｶﾗ ｼﾀ ?｣
D292  20__        JR   NZ, ｢ｼﾀ ﾉ ﾏﾏ｣ ﾍ
D294  3E97        LD   A,97H               ;A <── ┐ ﾉ ｺｰﾄﾞ
D296  18__        JR   ｼﾀ･ｷｾｷ ﾍ
D298  3E91        LD   A,91H               ;｢ｼﾀ ﾉ ﾏﾏ｣ ･･ A <── │ ﾉ ｺｰﾄﾞ
D29A  320EE0      LD   (0E00EH),A          ;ｼﾀ･ｷｾｷ ﾉ ｼｮﾘ  ※※ E00EH = ORBIT ※※
D29D  3A0DE0      LD   A,(0E00DH)          ; ※※ E00DH = TS ※※
D2A0  FE00        CP   00H                 ;ﾄﾛﾝ ｶ ?
D2A2  20__        JR   NZ, ｻｰｸ･ｼﾀ･ﾊﾞｲｸ ﾍ

D2A4  3E6E        LD   A,6EH               ;ﾄﾛﾝ･ｼﾀ･ﾊﾞｲｸ  ･･ 6EH = 110
D2A6  320FE0      LD   (0E00FH),A          ;BIKE ﾆ 110 ｦ ｶｸﾉｳ ｽﾙ
D2A9  3E02        LD   A,02H               ; ※※ 2 = ｼﾀ ﾉ ｺｰﾄﾞ ※※
D2AB  3204E0      LD   (0E004H),A          ;TRMV ﾆ 2 ｦ ｶｸﾉｳ ｽﾙ
D2AE  C9          RET

D2AF  3ED2        LD   A,0D2H              ;ｻｰｸ･ｼﾀ･ﾊﾞｲｸ  ･･ D2H = 210
D2B1  320FE0      LD   (0E00FH),A          ;BIKE ﾆ 210 ｦ ｶｸﾉｳ ｽﾙ
D2B4  3E02        LD   A,02H               ; ※※ 2 = ｼﾀ ﾉ ｺｰﾄﾞ ※※
D2B6  3205E0      LD   (0E005H),A          ;SKMV ﾆ 2 ｦ ｶｸﾉｳ ｽﾙ
D2B9  C9          RET
```

図4－25（CHRDN）

```
アドレス マシンコード      ニーモニック                 コメント

D2BA  3A0DE0     LD    A,(0E00DH)       ;ヒダリ・キャラクター (CHRLT)  ** E00DH = TS **
D2BD  FE00       CP    00H              ;トロン カ ?
D2BF  20__       JR    NZ, サーク・ヒダリ・キセキ ヘ
D2C1  3A04E0     LD    A,(0E004H)       ;トロン・ヒダリ・キセキ ノ ショリ  ** E004H = TRMV **
D2C4  18__       JR    「ウエ カラ ヒダリ ?」 ヘ
D2C6  3A05E0     LD    A,(0E005H)       ;サーク・ヒダリ・キセキ ノ ショリ  ** E005H = SKMV **

D2C9  FE08       CP    08H              ;「ウエ カラ ヒダリ ?」
D2CB  20__       JR    NZ, 「シタ カラ ヒダリ ?」 ヘ
D2CD  3E97       LD    A,97H            ;A <── ┐  ノ コード
D2CF  18__       JR    ヒダリ・キセキ ヘ
D2D1  FE02       CP    02H              ;「シタ カラ ヒダリ ?」
D2D3  20__       JR    NZ, 「ヒダリ ノ ママ」 ヘ
D2D5  3E98       LD    A,98H            ;A <── ┘  ノ コード
D2D7  18__       JR    ヒダリ・キセキ ヘ
D2D9  3E90       LD    A,90H            ;「ヒダリ ノ ママ」 ・・ A <── ─  ノ コード
D2DB  320EE0     LD    (0E00EH),A       ;ヒダリ・キセキ ノ ショリ  ** E00EH = ORBIT **
D2DE  3A0DE0     LD    A,(0E00DH)       ; ** E00DH = TS **
D2E1  FE00       CP    00H              ;トロン カ ?
D2E3  20__       JR    NZ, サーク・ヒダリ・バイク ヘ

D2E5  3E78       LD    A,78H            ;トロン・ヒダリ・バイク  ・・ 78H = 120
D2E7  320FE0     LD    (0E00FH),A       ;BIKE ニ 120 ヲ カクノウ スル
D2EA  3E04       LD    A,04H            ; ** 4 = ヒダリ ノ コード **
D2EC  3204E0     LD    (0E004H),A       ;TRMV ニ 4 ヲ カクノウ スル
D2EF  C9         RET

D2F0  3EDC       LD    A,0DCH           ;サーク・ヒダリ・バイク  ・・ DCH = 220
D2F2  320FE0     LD    (0E00FH),A       ;BIKE ニ 220 ヲ カクノウ スル
D2F5  3E04       LD    A,04H            ; ** 4 = ヒダリ ノ コード **
D2F7  3205E0     LD    (0E005H),A       ;SKMV ニ 4 ヲ カクノウ スル
D2FA  C9         RET
```

図4－26（CHRLT）

```
ｱﾄﾞﾚｽ ﾏｼﾝｺｰﾄﾞ        ﾆｰﾓﾆｯｸ                    ｺﾒﾝﾄ

D2FB  3A0DE0      LD   A,(0E00DH)          ;ﾐｷﾞ･ｷｬﾗｸﾀｰ (CHRRT)  ** E00DH = TS **
D2FE  FE00        CP   00H                 ;ﾄﾛﾝ ｶ ?
D300  20__        JR   NZ, ｻｰｸ･ﾐｷﾞ･ｷｾｷ ﾍ
D302  3A04E0      LD   A,(0E004H)          ;ﾄﾛﾝ･ﾐｷﾞ･ｷｾｷ ﾉ ｼｮﾘ  ** E004H = TRMV **
D305  18__        JR   ｢ｳｴ ｶﾗ ﾐｷﾞ ?｣ ﾍ
D307  3A05E0      LD   A,(0E005H)          ;ｻｰｸ･ﾐｷﾞ･ｷｾｷ ﾉ ｼｮﾘ  ** E005H = SKMV **

D30A  FE08        CP   08H                 ;｢ｳｴ ｶﾗ ﾐｷﾞ ?｣
D30C  20__        JR   NZ, ｢ｼﾀ ｶﾗ ﾐｷﾞ ?｣ ﾍ
D30E  3E9A        LD   A,9AH               ;A <── ┌ ﾉ ｺｰﾄﾞ
D310  18__        JR   ﾐｷﾞ･ｷｾｷ ﾍ
D312  FE02        CP   02H                 ;｢ｼﾀ ｶﾗ ﾐｷﾞ ?｣
D314  20__        JR   NZ, ｢ﾐｷﾞ ﾉ ﾏﾏ｣ ﾍ
D316  3E99        LD   A,99H               ;A <── └ ﾉ ｺｰﾄﾞ
D318  18__        JR   ﾐｷﾞ･ｷｾｷ ﾍ
D31A  3E90        LD   A,90H               ;｢ﾐｷﾞ ﾉ ﾏﾏ｣ ･･ A <── ─ ﾉ ｺｰﾄﾞ
D31C  320EE0      LD   (0E00EH),A          ;ﾐｷﾞ･ｷｾｷ ﾉ ｼｮﾘ  ** E00EH = ORBIT **
D31F  3A0DE0      LD   A,(0E00DH)          ; ** E00DH = TS **
D322  FE00        CP   00H                 ;ﾄﾛﾝ ｶ ?
D324  20__        JR   NZ, ﾄﾛﾝ･ﾐｷﾞ･ﾊﾞｲｸ ﾍ

D326  3E82        LD   A,82H               ;ﾄﾛﾝ･ﾐｷﾞ･ﾊﾞｲｸ ･･ 82H = 130
D328  320FE0      LD   (0E00FH),A          ;BIKE ﾆ 130 ｦ ｶｸﾉｳ ｽﾙ
D32B  3E06        LD   A,06H               ; ** 6 = ﾐｷﾞ ﾉ ｺｰﾄﾞ **
D32D  3204E0      LD   (0E004H),A          ;TRMV ﾆ 6 ｦ ｶｸﾉｳ ｽﾙ
D330  C9          RET

D331  3EE6        LD   A,0E6H              ;ｻｰｸ･ﾐｷﾞ･ﾊﾞｲｸ ･･ E6H = 230
D333  320FE0      LD   (0E00FH),A          ;BIKE ﾆ 230 ｦ ｶｸﾉｳ ｽﾙ
D336  3E06        LD   A,06H               ; ** 6 = ﾐｷﾞ ﾉ ｺｰﾄﾞ **
D338  3205E0      LD   (0E005H),A          ;SKMV ﾆ 6 ｦ ｶｸﾉｳ ｽﾙ
D33B  C9          RET
```

図4－27（CHRRT）

4－20／プリントルーチンPRINTの作成

表示ルーチンの最後に、下位サブルーチン　ＰＲＩＮＴ　を配置します。４－１８節で決めた仕様に従って書き下すと、リスト （図４－２８） のようになりました。

この段階で、今まで未定だった絶対アドレス、相対アドレスはすべて決定されているはずですから、空欄を記入して埋めておいて下さい。

図４－２８

```
■■ PRINT ( ﾌﾟﾘﾝﾄ ｻﾌﾞﾙｰﾁﾝ ) ■■

ｱﾄﾞﾚｽ ﾏｼﾝｺｰﾄﾞ     ﾆｰﾓﾆｯｸ                ｺﾒﾝﾄ

D33C  7A          LD    A,D             ;A <── ｷｬﾗｸﾀｰ ｼｷﾍﾞﾂ ｺｰﾄﾞ
D33D  FE01        CP    01H             ;ｷｬﾗｸﾀｰ ﾊ ﾊﾞｲｸ ｶ ?
D33F  20__        JR    NZ, ｷｾｷ･ｱﾄﾘﾋﾞｭｰﾄ ﾍ
D341  3E27        LD    A,27H           ;ﾊﾞｲｸ･ｱﾄﾘﾋﾞｭｰﾄ  ･･ COLOR 7,CGEN 1
D343  18__        JR    ｱﾄﾘﾋﾞｭｰﾄ･ｼｭﾂﾘｮｸ ﾍ
D345  3A0DE0      LD    A,(0E00DH)      ;ｷｾｷ･ｱﾄﾘﾋﾞｭｰﾄ ･･ ※※ E00DH = TS ※※
D348  FE00        CP    00H             ;ﾄﾛﾝ ｶ ?
D34A  20__        JR    NZ, ｻｰｸ･ｱﾄﾘﾋﾞｭｰﾄ ﾍ
D34C  7A          LD    A,D             ;ﾄﾛﾝ･ｱﾄﾘﾋﾞｭｰﾄ ･･ A <── ｷｬﾗｸﾀｰ ｼｷﾍﾞﾂ ｺｰﾄﾞ
D34D  FE00        CP    00H             ;ｷｾｷ ｶ ?
D34F  20__        JR    NZ, ﾄﾛﾝ･ｼｮｳﾄﾂ ﾍ
D351  3E26        LD    A,26H           ;ﾄﾛﾝ･ｷｾｷ･ｱﾄﾘﾋﾞｭｰﾄ  ･･ COLOR 6, CGEN 1
D353  18__        JR    ｱﾄﾘﾋﾞｭｰﾄ･ｼｭﾂﾘｮｸ ﾍ
D355  3E06        LD    A,06H           ;ﾄﾛﾝ･ｼｮｳﾄﾂ  ･･ COLOR 6, CGEN 0
D357  18__        JR    ｱﾄﾘﾋﾞｭｰﾄ･ｼｭﾂﾘｮｸ ﾍ
D359  7A          LD    A,D             ;ｻｰｸ･ｱﾄﾘﾋﾞｭｰﾄ  ･･ A <── ｷｬﾗｸﾀｰ ｼｷﾍﾞﾂ ｺｰﾄﾞ
D35A  FE00        CP    00H             ;ｷｾｷ ｶ ?
D35C  20__        JR    NZ, ｻｰｸ･ｼｮｳﾄﾂ ﾍ
D35E  3E22        LD    A,22H           ;ｻｰｸ･ｷｾｷ･ｱﾄﾘﾋﾞｭｰﾄ  ･･ COLOR 2, CGEN 1
D360  18__        JR    ｱﾄﾘﾋﾞｭｰﾄ･ｼｭﾂﾘｮｸ ﾍ
D362  3E02        LD    A,02H           ;ｻｰｸ･ｼｮｳﾄﾂ  ･･ COLOR 2, CGEN 0

D364  CBA0        RES   4,B             ;ｱﾄﾘﾋﾞｭｰﾄ･ｼｭﾂﾘｮｸ  ･･ BC = ｱﾄﾘﾋﾞｭｰﾄ VRAM ｱﾄﾞﾚｽ
D366  ED79        OUT   (C),A           ;I/O(BC) <── A  ･･ A = ｱﾄﾘﾋﾞｭｰﾄ･ｺｰﾄﾞ
D368  CBE0        SET   4,B             ;ｷｬﾗｸﾀｰ･ｼｭﾂﾘｮｸ  ･･ BC = ﾃｷｽﾄ VRAM ｱﾄﾞﾚｽ
D36A  7B          LD    A,E             ;A <── ｷｬﾗｸﾀｰ ﾉ ASCII ｺｰﾄﾞ
D36B  ED79        OUT   (C),A           ;I/O(BC) <── A
D36D  C9          RET
```

４−21／画面反転サブルーチン(CREV)の配置

いよいよマシン語プログラムの作成も最後を迎えました。ＰＲＩＮＴサブルーチンは、Ｄ３６ＤＨ番地で終りましたから、このすぐ後　Ｄ３６ＥＨ　番地から、画面反転サブルーチンを格納いたします。これはすでに前章で完成したものを利用いたします。リロケータブルになっていましたから、そのままでいいですね。

リスト図４−２９

```
■■ CREV ( ｶﾞﾒﾝ ﾊﾝﾃﾝ ｻﾌﾞﾙｰﾁﾝ ) ■■

ｱﾄﾞﾚｽ ﾏｼﾝｺｰﾄﾞ      ﾆｰﾓﾆｯｸ           ｺﾒﾝﾄ

D36E  010020      LD   BC,2000H
D371  ED78        IN   A,(C)       ;ﾙｰﾌﾟ
D373  CBDF        SET  3,A
D375  ED79        OUT  (C),A
D377  03          INC  BC
D378  78          LD   A,B
D379  FE23        CP   23H
D37B  38F4        JR   C, ﾙｰﾌﾟ ﾍ
D37D  79          LD   A,C
D37E  FEE8        CP   0E8H
D380  38EF        JR   C, ﾙｰﾌﾟ ﾍ
D382  C9          RET
```

以上をもちまして、すべてのサブルーチンが完成いたしました。本当に御苦労様でした。

☕コーヒーブレイク

マシン語プログラムの作成過程がわかるように記述したつもりですが、雰囲気は伝わったでしょうか？

この程度の長さのプログラムでも、ハンド・アセンブルですと、少しウンザリするというのが実感だと思います。特に相対アドレスの計算は面倒ですね。

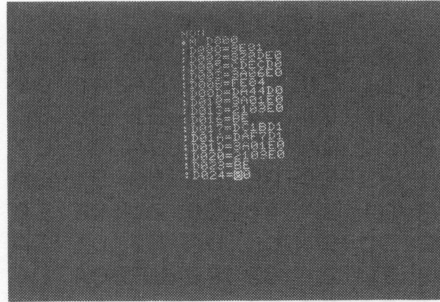
X１用のアセンブラがあったらなあ、とは誰もが考える所ですが、私の知る限りまだ、満足すべきアセンブラは発表されていないようです。

唯一、『I／O』誌の１９８３年3月号に　dB　SOFT　から「ミニアセンブラ」が発表されております。このアセンブラはBASICのREM文でニーモニックを入力し、マシン語アセンブラサブルーチンをCALLすることでアセンブルする形のものですが、書式が少し違う点、大きなプログラムが組めない点、時々原因不明のアセンブル・エラーを生ずる点など、まだまだ満足できるものではありません。1ユーザーとして、オールマシン語版のエディタ・アセンブラを望む所です。

けれど…　不満を言っても今はこれしかないのだから！　と思い直して、私はトロン・ゲームのマシン語部分を「ミニアセンブラ」で作成してみたのです。こわごわだったのですが、幸いなことに何とか動いてくれて、ニーモニックはキチンとマシンコードに変換されました。参考までに、私の作成した「ミニアセンブラ」用アセンブルリストを付録5に挙げておきましたので、御覧下さい。

4－22／チエック・サムの使い方

さて、プログラムが書き上がりましたので、いよいよ、モニターを起動して、メモリーに書き込んでゆきます。ニーモニック、コメントは無視して、アドレスを参照しながら、＊M　コマンドにより、マシンコードを入力して行って下さい。

図4－30

D000H番地から、D382H番地まで入力終了しましたか？

これが終わりましたら、取りあえずテープにセーブしておいて下さい。＊S　コマンドで、

＊S　D000　D382　D000　：TRON　MAS

とでもすれば、D000H〜D382H番地のメモリー内容がテープにセーブされます。

マシン語プログラムの場合、入力ミスエラーがあっても、システムはエラー表示をしてくれず、テスト実行でも「暴走」によりプログラム本体を壊すことがありますので、一応セーブしておく心掛けは、BASICプログラム以上に大切です。

さて、セーブが終了しましたら、入力ミスのチェックに移ります。これは、自分の作成したプログラムの場合は大変で、プログラムを小部分に分け、テストプログラムを通して実行確認していかねばなりません。本当は、開発の雰囲気を伝えるのに、部分部分のテストプログラムを掲載して読者の皆様にも実験していただくとよいのですが、ページ数が莫大になりそうなので、ここでは省略します。

かわりに、すでに完成品がどこかにあるとして、それと比較してミスチェックをする方法をとりましょう。マイコン雑誌のマシン語プログラムでは大ていこの方法をとっていますね。この時、必ず登場するものに、チェックサム（Check　Sum）という表があります。　次に掲げるのが、私たちが作成したトロンゲームのマシン語プログラム部分のチェックサムです。（図4－31①〜図4－31④）

図4－31①

```
:D000 = 3E 01 32 0D E0 CD EC D0  : E7
:D008 = 3A 06 E0 FE 04 DA 44 D0  : 10
:D010 = 3A 01 E0 21 03 E0 BE DC  : B9
:D018 = 1B D1 DA F7 D1 3A 01 E0  : A9
:D020 = 21 03 E0 BE C4 58 D1 DA  : 89
:D028 = F7 D1 3A 00 E0 21 02 E0  : E5
:D030 = BE DC 91 D1 DA F7 D1 3A  : D8
:D038 = 00 E0 21 02 E0 BE C4 C4  : 29
:D040 = D1 DA F7 D1 CD 1B D1 DA  : 06
:D048 = F7 D1 CD 58 D1 DA F7 D1  : 60
:D050 = CD 91 D1 DA F7 D1 CD C4  : 62
:D058 = D1 DA F7 D1 3A 0C E0 FE  : 97
:D060 = 08 20 05 CD 2E D1 18 15  : 26
:D068 = FE 02 20 05 CD 69 D1 18  : 44
:D070 = 0C FE 04 20 05 CD 9F D1  : 70
:D078 = 18 03 CD D2 D1 3E 01 32  : FC
-----------------------------------------
        33 A2 1A 4C B6 06 55 B1  : FD

:D080 = 07 E0 C3 F7 D1 3E 00 32  : E2
:D088 = 0D E0 CD EC D0 3A 04 E0  : 94
:D090 = 21 0C E0 86 FE 0A 20 03  : BE
:D098 = 21 04 E0 7E FE 08 20 10  : B9
:D0A0 = CD 1B D1 38 08 CD 2E D1  : C5
:D0A8 = 3E 01 32 07 E0 C3 F7 D1  : E3
:D0B0 = FE 02 20 10 CD 58 D1 38  : 5E
:D0B8 = 08 CD 69 D1 3E 01 32 07  : 87
:D0C0 = E0 C3 F7 D1 FE 04 20 10  : 9D
:D0C8 = CD 91 D1 38 08 CD 9F D1  : AC
:D0D0 = 3E 01 32 07 E0 C3 F7 D1  : E3
:D0D8 = FE 06 20 BC CD C4 D1 38  : 7A
:D0E0 = 08 CD D2 D1 3E 01 32 07  : F0
:D0E8 = E0 C3 F7 D1 3A 0D E0 FE  : 90
:D0F0 = 00 20 0C 7E 32 0C E0 21  : E9
:D0F8 = 00 E0 56 23 5E 18 0C 3A  : 15
-----------------------------------------
        38 A6 21 16 4B FD F1 50  : 9E
```

図4－31②

```
:D100 = 05 E0 32 0C E0 21 02 E0  : 06
:D108 = 56 23 5E 21 00 30 4A 16  : 88
:D110 = 00 06 28 19 10 FD 09 22  : 7F
:D118 = 08 E0 C9 2A 08 E0 11 28  : FC
:D120 = 00 B7 ED 52 44 4D ED 78  : EC
:D128 = FE 20 28 0B B7 C9 2A 08  : 03
:D130 = E0 11 28 00 B7 ED 52 22  : 31
:D138 = 0A E0 3E 08 32 0C E0 3A  : 88
:D140 = 0D E0 FE 00 20 09 3A 01  : 4F
:D148 = E0 3D 32 01 E0 18 07 3A  : 89
:D150 = 03 E0 3D 32 03 E0 37 C9  : 35
:D158 = 2A 08 E0 11 28 00 19 44  : A8
:D160 = 4D ED 78 FE 20 28 09 B7  : B8
:D168 = C9 2A 08 E0 11 28 00 19  : 2D
:D170 = 22 0A E0 3E 02 32 0C E0  : 6A
:D178 = 3A 0D E0 FE 00 20 09 3A  : 88
-----------------------------------------
        D7 E4 89 33 3A E0 5E 4E  : 3D


:D180 = 01 E0 3C 32 01 E0 18 07  : 4F
:D188 = 3A 03 E0 3C 32 03 E0 37  : A5
:D190 = C9 2A 08 E0 2B 44 4D ED  : 84
:D198 = 78 FE 20 28 06 B7 C9 2A  : 6E
:D1A0 = 08 E0 2B 22 0A E0 3E 04  : 61
:D1A8 = 32 0C E0 3A 0D E0 FE 00  : 43
:D1B0 = 20 09 3A 00 E0 3D 32 00  : B2
:D1B8 = E0 18 07 3A 02 E0 3D 32  : 8A
:D1C0 = 02 E0 37 C9 2A 08 E0 23  : 17
:D1C8 = 44 4D ED 78 FE 20 28 06  : 42
:D1D0 = B7 C9 2A 08 E0 23 22 0A  : E1
:D1D8 = E0 3E 06 32 0C E0 3A 0D  : 89
:D1E0 = E0 FE 00 20 09 3A 00 E0  : 21
:D1E8 = 3C 32 00 E0 18 07 3A 02  : A9
:D1F0 = E0 3C 32 02 E0 37 C9 CD  : FD
:D1F8 = 26 D2 ED 4B 08 E0 16 00  : 2E
-----------------------------------------
        B5 8A 03 D4 7A 3E 36 7A  : 7E
```

図4－31③

```
:D200 = 3A 0E E0 5F CD 3C D3 3A  : 9D
:D208 = 07 E0 FE 00 20 0D ED 4B  : 4A
:D210 = 0A E0 16 01 3A 0F E0 5F  : 89
:D218 = C3 3C D3 ED 4B 0A E0 16  : 0A
:D220 = 02 1E E8 C3 3C D3 3A 0C  : 20
:D228 = E0 FE 02 CA 79 D2 FE 04  : F7
:D230 = CA BA D2 FE 06 CA FB D2  : F1
:D238 = 3A 0D E0 FE 00 20 05 3A  : 84
:D240 = 04 E0 18 03 3A 05 E0 FE  : 1C
:D248 = 04 20 04 3E 99 18 0A FE  : 1F
:D250 = 06 20 04 3E 98 18 02 3E  : 58
:D258 = 91 32 0E E0 3A 0D E0 FE  : D6
:D260 = 00 20 0B 3E 64 32 0F E0  : EE
:D268 = 3E 08 32 04 E0 C9 3E C8  : 2B
:D270 = 32 0F E0 3E 08 32 05 E0  : 7E
:D278 = C9 3A 0D E0 FE 00 20 05  : 13
---------------------------------------
        CC B0 BB 95 1C 60 F6 DB  : 19


:D280 = 3A 04 E0 18 03 3A 05 E0  : 58
:D288 = FE 04 20 04 3E 9A 18 0A  : 20
:D290 = FE 06 20 04 3E 97 18 02  : 17
:D298 = 3E 91 32 0E E0 3A 0D E0  : 16
:D2A0 = FE 00 20 0B 3E 6E 32 0F  : 16
:D2A8 = E0 3E 02 32 04 E0 C9 3E  : 3D
:D2B0 = D2 32 0F E0 3E 02 32 05  : 6A
:D2B8 = E0 C9 3A 0D E0 FE 00 20  : EE
:D2C0 = 05 3A 04 E0 18 03 3A 05  : 7D
:D2C8 = E0 FE 08 20 04 3E 97 18  : F7
:D2D0 = 0A FE 02 20 04 3E 98 18  : 1C
:D2D8 = 02 3E 90 32 0E E0 3A 0D  : 37
:D2E0 = E0 FE 00 20 0B 3E 78 32  : F1
:D2E8 = 0F E0 3E 04 32 04 E0 C9  : 10
:D2F0 = 3E DC 32 0F E0 3E 04 32  : AF
:D2F8 = 05 E0 C9 3A 0D E0 FE 00  : D3
---------------------------------------
        27 E6 94 17 17 B2 6C AD  : 9A
```

図４－３１④

```
:D300 = 20 05 3A 04 E0 18 03 3A  : 98
:D308 = 05 E0 FE 08 20 04 3E 9A  : E7
:D310 = 18 0A FE 02 20 04 3E 99  : 1D
:D318 = 18 02 3E 90 32 0E E0 3A  : 42
:D320 = 0D E0 FE 00 20 0B 3E 82  : D6
:D328 = 32 0F E0 3E 06 32 04 E0  : 7B
:D330 = C9 3E E6 32 0F E0 3E 06  : 52
:D338 = 32 05 E0 C9 7A FE 01 20  : 79
:D340 = 04 3E 27 18 1F 3A 0D E0  : C7
:D348 = FE 00 20 0D 7A FE 00 20  : C3
:D350 = 04 3E 26 18 0F 3E 06 18  : EB
:D358 = 0B 7A FE 00 20 04 3E 22  : 07
:D360 = 18 02 3E 02 CB A0 ED 79  : 2B
:D368 = CB E0 7B ED 79 C9 01 00  : 56
:D370 = 20 ED 78 CB DF ED 79 03  : 98
:D378 = 78 FE 23 38 F4 79 FE E8  : 24
-----------------------------------------
        1B E6 D7 06 E0 92 96 CD  : B3
```

```
:D380 = 38 EF C9 00 00 00 00 00  : F0
:D388 = 00 00 00 00 00 00 00 00  : 00
:D390 = 00 00 00 00 00 00 00 00  : 00
:D398 = 00 00 00 00 00 00 00 00  : 00
:D3A0 = 00 00 00 00 00 00 00 00  : 00
:D3A8 = 00 00 00 00 00 00 00 00  : 00
:D3B0 = 00 00 00 00 00 00 00 00  : 00
:D3B8 = 00 00 00 00 00 00 00 00  : 00
:D3C0 = 00 00 00 00 00 00 00 00  : 00
:D3C8 = 00 00 00 00 00 00 00 00  : 00
:D3D0 = 00 00 00 00 00 00 00 00  : 00
:D3D8 = 00 00 00 00 00 00 00 00  : 00
:D3E0 = 00 00 00 00 00 00 00 00  : 00
:D3E8 = 00 00 00 00 00 00 00 00  : 00
:D3F0 = 00 00 00 00 00 00 00 00  : 00
:D3F8 = 00 00 00 00 00 00 00 00  : 00
-----------------------------------------
        38 EF C9 00 00 00 00 00  : F0
```

チェックサムの形式はいろいろあって、雑誌によっても異なりますが、本書で採用したのは、ＢＡＳＩＣのモニターのダンプ形式を真似た体裁です。欄外の右にあるのは、横一列（８バイト分）の和の１６進下２桁を表示しており、下にあるのは、縦一列（１６バイト分）の和の１６進下２桁を表示しています。右下隅のは、トータルサムといって、８×１６＝１２８バイト分の総和の下２桁を意味しています。

この表を活用するためには、読者の入力したマシン語部分について、チェックサムを表示（できれば同一形式で！）しなくてはなりません。

付録３に、「チェックサムプログラム」を載せておきましたので、入力してみるとよいと思います。自前のものですが、なかなか便利ですよ！

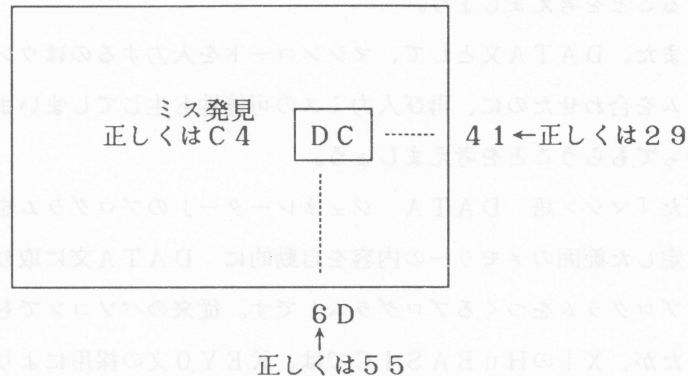
さて、「チェックサムプログラム」が完成しましたら、先程テープにセーブしておいたトロンゲームのマシン語部分をロードいたします。念のため、 ＣＬＥＡＲ　＆ＨＤ０００ を実行しておいた方が安全です。しかる後に、　ＬＯＡＤＭ　ないしは、モニター起動後の　＊Ｌコマンド　によりマシン語をロードして下さい。

さあ準備完了です。チェックサムプログラムをＲＵＮして下さい。すべての注意は画面に表示されますから、指示通りにキーを操作して下さい。開始アドレス＝Ｄ０００，終了アドレス＝Ｄ３８２　と指定し、リターン・キーを押すと、最初の　１２８バイト分のチェックサム画面が現われます。

図４－３２

```
:D000 = 3E 01 32 0D E0 CD EC D0  : E7
:D008 = 3A 06 E0 FE 04 DA 44 D0  : 10
:D010 = 3A 01 E0 21 03 E0 BE DC  : B9
:D018 = 1B D1 DA F7 D1 3A 01 E0  : A9
:D020 = 21 03 E0 BE C4 58 D1 DA  : 89
:D028 = F7 D1 3A 00 E0 21 02 E0  : E5
:D030 = BE DC 91 D1 DA F7 D1 3A  : D8
:D038 = 00 E0 21 02 E0 BE DC C4  : 41
:D040 = D1 DA F7 D1 CD 1B D1 DA  : 06
:D048 = F7 D1 CD 58 D1 DA F7 D1  : 60
:D050 = CD 91 D1 DA F7 D1 CD C4  : 62
:D058 = D1 DA F7 D1 3A 0C E0 FE  : 97
:D060 = 08 20 05 CD 2E D1 18 15  : 26
:D068 = FE 02 20 05 CD 69 D1 18  : 44
:D070 = 0C FE 04 20 05 CD 9F D1  : 70
:D078 = 18 03 CD D2 D1 3E 01 32  : FC
-----------------------------------------
        33 A2 1A 4C B6 06 6D B1  : 15
```

さて、仮に、最初のチェックサムが図４－３２のようであったとしましょう。まず、トータルサムを見ると違っていますね。つまり、どこかにミスがあるのです。

次にその位置の探し方ですが、縦の和を見ていくと、左から７列目の和が異なっています。これは、ミス位置がこの列内にあることを意味しています。続いて横の和を見ます。上から８行目が違っています。こうして、ミス位置は、次のようにわかります。

ミスの位置が判明したら、指示に従って訂正をして下さい。この例の場合、ミスのあるアドレスは、Ｄ０３ＥＨ番地です。　Ｄ０３Ｅ　とキー・インすると、

Ｄ０３Ｅ　＝　ＤＣ　→　？
現在の内容

と尋ねてきますから、正しい値　Ｃ４　を入力して下さい。すると、チェックサムプログラムは再びチェックサムを計算し表示してきます。直っているのを確認したら、リターンキーを押すと、次の１２８バイトのチェックサムに進みます。

このように「対話型」の方法で、チェックサムがきちんと合うまで操作を続けます。

こうしてすべてが終了すると、正しいプログラムが完成したことになりますから、再びテープにマシン語部分（Ｄ０００Ｈ～Ｄ３８２Ｈ）をセーブしておいて下さい。

［注］　トータルサム、横の和、縦の和がすべて一致しても、まだミスが残る可能性はあります。しかし、注意深く入力すれば、現実には、このようなことはまず起りませんから御安心下さい。

４－23／DATA文に直してアスキーセーブ！

さて、この後、ＢＡＳＩＣのメインルーチンを作ればゲームは完成しますが、このままでは、後日ゲームをする時、まず、ＢＡＳＩＣ部分をロード、次にマシン語部分をロードという具合に２度手間になってしまいますね。

そこで、ＤＡＴＡ文を用いて、マシン語プログラムをＢＡＳＩＣ本体内に取り込み、プログラムを１本化することを考えましょう。

しかし、またまた、ＤＡＴＡ文として、マシンコードを入力するのはウンザリですね。せっかくチェックサムを合わせたのに、再び入力ミスの可能性も生じてしまいます。そこで、この作業をＸ１にやってもらうことを考えましょう。

付録４に掲げた「マシン語　ＤＡＴＡ　ジェネレーター」のプログラムを御覧下さい。このプログラムは指定した範囲のメモリーの内容を自動的に　ＤＡＴＡ文に取り込んで、行番号つきで生成する「プログラムをつくるプログラム」です。従来のパソコンでも、この種のプログラムはありましたが、Ｘ１のＨｕＢＡＳＩＣでは、ＫＥＹ０文の採用により簡単に作成することができます。ＫＥＹ０文はこの他にも「ユーザー定義文字作成プログラム」などにも応用される有用な命令ですので、本プログラムを参考に他の応用を考えてみて下さい。

さて、「マシン語　ＤＡＴＡ　ジェネレーター」が入力セーブを終わり、完成しているといたします。　ＣＬＥＡＲ　＆ＨＤ０００　の後、できあがっているトロンゲームのマシン語部分をロードし、ＤＡＴＡジェネレーターをＲＵＮいたします。指示通りにキーを操作すると、約４０秒で、トロンゲームのマシン語部分がＤＡＴＡ文として完成いたします。コンピューターの威力ってすごいですね。（本体プログラムがすでにセーブしてあるとして！）しかる後に、 ＤＥＬＥＴＥ－９００ により「ＤＡＴＡジェネレーター」の本体を消去すると、行番号３００００から、見事ＤＡＴＡ文だけが残りますね。

これをテープにセーブすればよいのですが、後にゲームメイン部分とドッキングさせることを前提に、アスキーセーブというのを試みてみましょう。この言葉、御存知でしたか？

ＢＡＳＩＣのプログラムがメモリーに格納される時は、省メモリー化のために「中間コード」というものが採用されます。たとえば、　ＧＯＴＯ　の中間コードは、　８０Ｈ　です。何故こんなことをするかというと、　ＧＯＴＯ　のまま格納すると、４文字ですから４バイト

分メモリーを食います。しかし、中間コードで　８０Ｈ　ならたった１バイトで済んでしまいますね。ＢＡＳＩＣの各コマンドには、すべてこのような「中間コード」が決められていて、メモリーにはこの形式で格納されており、普通にテープにセーブする時も、この形式でテープに録音されます。

ところが、２つのＢＡＳＩＣプログラムを１本に合わせたりする時には、都合の悪いことがおきます。ＢＡＳＩＣインタプリターは、どこからどこまでが１行で、次の行はどこから始まるかを判断できずに、デタラメなプログラムと見なしてしまうからです。そこで、このような目的のために、アスキーセーブがあるのです。

ただのセーブではなく、アスキー形式でプログラムをセーブすると、プログラムのリスト通り　――　つまり「中間コード」を用いずに　――　にテープにセーブされるのです。バイト数が多くなりますから、時間がかかりますが、後に２本のプログラムを１本にまとめる（マージするといいます）ためには、是非アスキーセーブをしておいて下さい。

では、どうしたら普通のセーブではなく、アスキーセーブができるのでしょう。私もそうでしたが、初心者の方はつい見逃がす所だと思います。マニュアル「３４ページ」を御覧下さい。ここに、小さく出ているのです（私も最初気づかなかった位です！）。

すなわち、ＢＡＳＩＣプログラムをアスキーセーブするには、

```
          ┌── ファイル名（何でもよい）
          ↓
SAVE "TRON DATA", A ←┐
                     │
          アスキーセーブの指示
```

とすればよいのです。

まだ御存知なかった方は、是非試してみて下さい。何回もテープが止まっては動くのを繰り返して、トロンＤＡＴＡの場合、約１分かかってセーブされるはずです。

４−24／トロンゲームマシン語版の完成！

さて、いよいよ長い道のりも大詰めを迎えました。ＢＡＳＩＣの本体プログラムの作成です。すでに、オールＢＡＳＩＣ版は完成しておりますので、少しの変更で済みます。

以下、変更点を述べますので、オールＢＡＳＩＣ版リストを参照しながらお読み下さい。

変更点①　初期化ルーチン中、最初の１３０行でマシン語フリーエリアの確保をしておきます。すなわち、　ＣＬＥＡＲ　＆ＨＤ０００　を加えます。

図４－３３

```
110 ' ■■■■    ｼｮｷｶ    ■■■■
120 '
130 CLEAR &HD000 :INIT :WIDTH 40 :CLS 4 :CLICK OFF :TEMPO 200
140 DEFINT A-Z
150 '
160 GOSUB "ｵｰﾌﾟﾆﾝｸﾞ"
170 GOSUB "ｷｬﾗｸﾀｰ･ﾃｲｷﾞ"
```

変更点②　１９０行～２５０行の数値の初期設定は全面的に書き直して、マシン語を書き込むルーチンにします。

図４－３４

```
190 ' ■■■■    ﾏｼﾝｺﾞ･ｻﾌﾞﾙｰﾁﾝ ﾉ ｶｷｺﾐ    ■■■■
200 '
210 RESTORE 30000 :ADR=&HD000
220 FOR I=0 TO &H382
230   READ MC$ :POKE ADR+I,VAL("&H"+MC$)
240 NEXT
250 DEF USR=&HD085      :'<—— TRON MOVE ﾙｰﾁﾝ
```

ＲＥＡＤ文とＰＯＫＥ文の使い方、よろしいですか？　また、２５０行でＵＳＲ関数の定義をしておきました。

変更点③　１１５０行～１２８０行の出発位置決定ルーチンは、次のように３つのルーチンに分けて書き直してみました。

図４－３５

```
1150 ' ■■■■    ｼｭｯﾊﾟﾂ ｲﾁ ﾉ ｹｯﾃｲ    ■■■■
1160 '
1170 RANDOMIZE  TIME-20864
1180 X=INT(RND*20)+10
1190 Y=INT(RND*13)+6
1200 REPEAT
1210   U=INT(RND*36)+2
1220   V=INT(RND*19)+3
1230   UNTIL  (X-U)*(X-U)+(Y-V)*(Y-V)>=25
1240 DUV=INT(RND*4)*2+2    :'<—— SARK ｲﾄﾞｳ･ｺｰﾄﾞ  (2,4,6,8)
1250 '
1260 ' ■■■■    ﾏｼﾝｺﾞ ﾜｰｸ･ｴﾘｱ ｼｮｷ ｾｯﾃｲ    ■■■■
1270 '
1280 POKE &HE000,X,Y              :' E000H=X,  E001H=Y
1290 POKE &HE002,U,V              :' E002H=U,  E003H=V
1300 POKE &HE004,4                :' E004H=DX,DY <— TRON ｲﾄﾞｳ･ｺｰﾄﾞ  (2,4,6,8)
1310 POKE &HE005,DUV              :' E005H=DU,DV <— SARK ｲﾄﾞｳ･ｺｰﾄﾞ
1320 POKE &HE007,0                :' E007H=Cont flag (0=Cont, 1=End)
1360 '
1370 ' ■■■■    ｼｮｷ ﾋｮｳｼﾞ    ■■■■
1380 '
1500 CGEN 1 :COLOR 7
1510 LOCATE X,Y :PRINT#0 CHR$(120)
1520 LOCATE U,V :PRINT#0 CHR$(220) :CGEN 1
1530 FOR I=1 TO 3 :BEEP :NEXT
```

特に大切なのは、１２６０行～１３２０行のマシン語ワークエリアの初期設定の部分です。出発位置をＢＡＳＩＣプログラムで決定したら、最初にサークが判断する時のために、ワークエリアに必要なデータを書き込んでおく必要があります。旧ＢＡＳＩＣ版で使われていた　ＤＸ，ＤＹ，ＤＵ，ＤＶ　という変数は使用せず、移動方向コードに統一します。また、トロンの初期方向は左（コード４）に決めました。

変更点④　２０００行～２５５０行のメインループは完全に書き換えてしまいます。乱数を書き込み、サーク移動ルーチン（Ｄ０００Ｈ）を呼び出し、続行判定フラグをチェックし、次に、テンキー値をもって、トロン移動ルーチン（Ｄ０８５Ｈ）を呼び出し、続行判定フラグをチェックし、ループに戻るという構成になっています。

図４－３６

```
2000 ' ■■■■   ﾒｲﾝ･ﾙｰﾌﾟ   ■■■■
2010 '
2020 ' ────   SARK ﾉ ｲﾄﾞｳ   ────
2030 '
2035 POKE &HE006,INT(RND*100)        :'<─── E006H=Random
2040 CALL &HD000                     :'<─── SARK MOVE ﾙｰﾁﾝ
2050 '
2060 F=PEEK(&HE007) :IF F=0 THEN 2110  :'<─── Cont flag ﾉ ﾁｪｯｸ
2070 T=T+1 :MUSIC "O4R3C1DEFGAB+CR3"
2080 GOSUB "ｶﾁ ﾉ ﾊﾝﾃｲ"
2090 IF H$<>"" THEN 3000  ELSE 1000
2100 '
2110 ' ────   TRON ﾉ ｲﾄﾞｳ   ────
2120 '
2130 I=STICK(0)
2140 A=USR(I)           :' USR=D085H  <─── TRON MOVE ﾙｰﾁﾝ
2150 '
2160 F=PEEK(&HE007) :IF F=0 THEN 2210  :'<─── Cont flag ﾉ ﾁｪｯｸ
2170 S=S+1 :MUSIC "O2R3+C1BAGFEDCR3"
2180 GOSUB "ｶﾁ ﾉ ﾊﾝﾃｲ"
2190 IF H$<>"" THEN 3000 ELSE 1000
2200 '
2210 FOR I=0 TO 100 :NEXT :GOTO 2000    :' ───> ﾒｲﾝ･ﾙｰﾌﾟ
```

オールＢＡＳＩＣ版に比べ、繁雑な条件判断をすべてマシン語ルーチンに任せたため、見やすい構造にできました。また、細かいことですが、効果音を変えてみました（２１７０行）。これも細かいことですが、乱数を０～９９の範囲にしました（２０３５行）。あと大切な点としては、２２１０行の　ＦＯＲ～ＮＥＸＴ　文です。これは何もしない空ループで、時間かせぎの役割をします。空ループなしでゲームをしてみるとわかりますが、マシン

語により高速になりすぎて、とても人間の反射神経では追いつかない位です。そこで、ＢＡＳＩＣ版では考えられないことですが、わざと遅くしています。速い方が好きな方は、ループ回数を１００より小さく、もう少し遅くしたい方は、１００より大きくして、各人に合うスピードに調節して下さい。

変更点⑤　３０２０行～３０６０行の画面反転の部分もマシン語化しましたね。これは次のように画面反転サブルーチン（Ｄ３６ＥＨ）を呼び出せば一発で済みます。

図４－３７

```
3000 ' ■■■■    ｹﾞｰﾑ ｵｰﾊﾞｰ    ■■■■
3010 '
3020 CALL &HD36E          :'<------ 1 ｶﾞﾒﾝ ﾊﾝﾃﾝ ﾙｰﾁﾝ
```

また、これも細かいことですが、旧ＢＡＳＩＣ版の３１００行、３１１０行の効果音も少し変えてみました。

図４－３８

```
3100 IF H$="TRON" THEN MUSIC "O5C2DEFGAB+C"
3110 IF H$="SARK" THEN MUSIC "O2+C4BAGFEDC"
```

変更点⑥　４０００行～４３６０行にわたる移動用サブルーチンは、マシン語部分に吸収しましたから消去いたします。

変更点⑦　先に、「ＤＡＴＡ　ジェネレーター」で作成済のマシン語部分を３００００行から組み入れます。このためには、ＤＡＴＡ文の入っているテープを頭出し状態にしておき、ＭＥＲＧＥ　というＢＡＳＩＣコマンドを実行いたします。ＤＡＴＡ文の部分は、アスキーセーブしてありましたから、マージが実行されて、１本のプログラムにまとめることができます。

以上が、変更点です。さて、ＤＡＴＡ文のマージは完了しましたか？　細部の体裁をととの

えて、次のようなリストが完成いたしました。テープに一応セーブして、ＲＵＮして下さい。

図４－３９

```
10 ' ████████████████████████████████████████
20 ' ██                                    ██
30 ' ██     TRON GAME  ver.2               ██
40 ' ██                                    ██
50 ' ██          by  Yasuhiro Shimizu      ██
60 ' ██                                    ██
70 ' ██          1983.10.15                ██
80 ' ██                                    ██
90 ' ████████████████████████████████████████
100 '
110 ' █████     ｼｮｷｶ     █████
120 '
130 CLEAR &HD000 :INIT :WIDTH 40 :CLS 4 :CLICK OFF :TEMPO 200
140 DEFINT A-Z
150 '
160 GOSUB "ｵｰﾌﾟﾆﾝｸﾞ"
170 GOSUB "ｷｬﾗｸﾀｰ･ﾃｲｷﾞ"
180 '
190 ' █████     ﾏｼﾝｺﾞ･ｻﾌﾞﾙｰﾁﾝ ﾉ ｶｷｺﾐ     █████
200 '
210 RESTORE 30000 :ADR=&HD000
220 FOR I=0 TO &H382
230   READ MC$ :POKE ADR+I,VAL("&H"+MC$)
240 NEXT
250 DEF USR=&HD085       :'<—— TRON MOVE ﾙｰﾁﾝ
260 '
270 ' █████     ｾﾂﾒｲ     █████
280 '
290 CLS
300 LOCATE 10,0 :COLOR 4 :CSIZE 3 :PRINT#0 "TRON GAME" :CSIZE 0
310 LOCATE 13,4 :COLOR 6 :PRINT "TRON: "
320 LOCATE 18,4 :COLOR 7 :CGEN 1 :PRINT CHR$(120) :CGEN 0
330 LOCATE 13,6 :COLOR 2 :PRINT "SARK: "
340 LOCATE 18,6 :COLOR 7 :CGEN 1 :PRINT CHR$(220) :CGEN 0
350 LOCATE 16,8  :PRINT "KEY"
360 LOCATE 17,10  :PRINT "8"
370 LOCATE 16,11 :PRINT "4 6"
380 LOCATE 17,12  :PRINT "2"
390 LOCATE 13,14 :COLOR 3 :PRINT "10 POINT MATCH"
400 LOCATE 13,18 :COLOR 7 :PRINT "HIT "
410 LOCATE 17,18 :CFLASH 1 :PRINT "RETURN KEY" :CFLASH 0
420 REPEAT
430   I$=INKEY$
440   UNTIL I$=CHR$(13)
450 '
460 ' █████     ｹﾞｰﾑ ｽﾀｰﾄ     █████
470 '
480 T=0 :S=0 :G=0
490 CLS
500 LOCATE 8,8 :COLOR 5 :CSIZE 3
510 PRINT#0 "GAME START" :CSIZE 0
520 MUSIC "O4R2C3DEFGAB+C"
530 '
1000 ' █████     ｶﾞﾒﾝ ﾂﾞｸﾘ     █████
1010 '
1020 CGEN 0 :CONSOLE 1,24 :CLS :CONSOLE
1030 COLOR 4
1040 LINE ( 0, 1)-(38, 1),"—"
1050 LINE ( 1,23)-(38,23),"—"
1060 LINE ( 0, 2)-( 0,22),"|"
1070 LINE (39, 2)-(39,22),"|"
1080 LOCATE  0, 1 :PRINT "┌"
1090 LOCATE 39, 1 :PRINT "┐"
1100 LOCATE  0,23 :PRINT "└"
1110 LOCATE 39,23 :PRINT "┘";
1120 LOCATE  1, 0 :COLOR 6 :PRINT "TRON:";
1130 LOCATE 20, 0 :COLOR 2 :PRINT "SARK:";
1140 '
1150 ' █████     ｼｭｯﾊﾟﾂ ｲﾁ ﾉ ｹｯﾃｲ     █████
1160 '
1170 RANDOMIZE  TIME-20864
1180 X=INT(RND*20)+10
1190 Y=INT(RND*13)+6
1200 REPEAT
1210   U=INT(RND*36)+2
1220   V=INT(RND*19)+3
1230   UNTIL (X-U)*(X-U)+(Y-V)*(Y-V)>=25
1240 DUV=INT(RND*4)*2+2    :'<—— SARK ｲﾄﾞｳ･ｺｰﾄﾞ (2,4,6,8)
1250 '
1260 ' █████     ﾏｼﾝｺﾞ ﾜｰｸ･ｴﾘｱ ｼｮｷ ｾｯﾃｲ     █████
1270 '
```

```
1280 POKE &HE000,X,Y              :' E000H=X,   E001H=Y
1290 POKE &HE002,U,V              :' E002H=U,   E003H=V
1300 POKE &HE004,4                :' E004H=DX,DY <── TRON ｲﾄﾞｳ･ｺｰﾄﾞ (2,4,6,8)
1310 POKE &HE005,DUV              :' E005H=DU,DV <── SARK ｲﾄﾞｳ･ｺｰﾄﾞ
1320 POKE &HE007,0                :' E007H=Cont flag (0=Cont, 1=End)
1360 '
1370 ' ■■■■■    ｼｮｷ ﾋｮｳｼﾞ    ■■■■■
1380 '
1500 CGEN 1 :COLOR 7
1510 LOCATE X,Y :PRINT#0 CHR$(120)
1520 LOCATE U,V :PRINT#0 CHR$(220) :CGEN 1
1530 FOR I=1 TO 3 :BEEP :NEXT
1540 '
2000 ' ■■■■■    ﾒｲﾝ･ﾙｰﾌﾟ    ■■■■■
2010 '
2020 ' ────    SARK ﾉ ｲﾄﾞｳ    ────
2030 '
2035 POKE &HE006,INT(RND*100)       :'<──── E006H=Random
2040 CALL &HD000                   :'<──── SARK MOVE ﾙｰﾁﾝ
2050 '
2060 F=PEEK(&HE007) :IF F=0 THEN 2110  :'<──── Cont flag ﾉ ﾁｪｯｸ
2070 T=T+1 :MUSIC "O4R3C1DEFGAB+CR3"
2080 GOSUB "ｶﾁ ﾉ ﾊﾝﾃｲ"
2090 IF H$<>"" THEN 3000  ELSE 1000
2100 '
2110 ' ────    TRON ﾉ ｲﾄﾞｳ    ────
2120 '
2130 I=STICK(0)
2140 A=USR(I)            :' USR=D085H  <──── TRON MOVE ﾙｰﾁﾝ
2150 '
2160 F=PEEK(&HE007) :IF F=0 THEN 2210  :'<──── Cont flag ﾉ ﾁｪｯｸ
2170 S=S+1 :MUSIC "O2R3+C1BAGFEDCR3"
2180 GOSUB "ｶﾁ ﾉ ﾊﾝﾃｲ"
2190 IF H$<>"" THEN 3000 ELSE 1000
2200 '
2210 FOR I=0 TO 100 :NEXT :GOTO 2000   :' ────> ﾒｲﾝ･ﾙｰﾌﾟ
2990 '
3000 ' ■■■■■    ｹﾞｰﾑｵｰﾊﾞｰ    ■■■■■
3010 '
3020 CALL &HD36E        :'<──── 1 ｶﾞﾒﾝ ﾊﾝﾃﾝ ﾙｰﾁﾝ
3070 '
3080 LOCATE 8,8 :COLOR 7 :CSIZE 2
3090 PRINT#0 " GAME!"+H$+" " :CSIZE 0
3100 IF H$="TRON" THEN MUSIC "O5C2DEFGAB+C"
3110 IF H$="SARK" THEN MUSIC "O2+C4BAGFEDC"
3120 '
3130 LOCATE 5,12 :PRINT "DO YOU WANT REPLAY [ Y or N ]"
3140 REPEAT
3150   I$=INKEY$
3160   UNTIL I$="Y" OR I$="N"
3170 '
3180 IF I$="Y" THEN 460       :' ────> ｹﾞｰﾑ ｽﾀｰﾄ
3190 INIT :CLS
3200 LOCATE 15,10 :PRINT "ｵﾂｶﾚｻﾏ ♥"
3210 END
3220 '
3980 ' *****   BASIC ｻﾌﾞﾙｰﾁﾝ   *****
3990 '
4980 ' ■■■■■■■■■■■■■■■
4990 '
5000   LABEL "ｶﾁ ﾉ ﾊﾝﾃｲ"
5010 '
5020 ' ■■■■■■■■■■■■■■■
5030 '
5040 GOSUB "ｽｺｱ"
5050 IF G=1 THEN 5140
5060 '
5070 H$=""
5080 IF T=10 THEN H$="TRON"
5090 IF S=10 THEN H$="SARK"
5100 RETURN
5110 '
5120 ' ────    ｼﾞｭｰｽ ﾉ ｼｮﾘ    ────
5130 '
5140 H$=""
5150 IF T=S+2 THEN H$="TRON"
5160 IF S=T+2 THEN H$="SARK"
5170 RETURN
5180 '
5190 '
5980 ' ■■■■■■■■■■
5990 '
6000   LABEL "ｽｺｱ"
6010 '
6020 ' ■■■■■■■■■■
6030 '
6040 MUSIC "O5G1+C" :CGEN 0
6050 LOCATE 8,0 :COLOR 6
```

```
6060 PRINTUSING "##";T;
6070 LOCATE 27,0 :COLOR 2
6080 PRINTUSING "##";S;
6090 '
6100 LOCATE 12,0 :PRINT "      ";
6110 IF T>=9 AND S>=9 AND T=S THEN G=1 :LOCATE 12,0 :COLOR 7 :CFLASH 1 :PRINT "ｼﾞｭｰｽ"; :CFLASH 0
 :FOR I=1 TO 2 :BEEP 1 :PAUSE 5 :BEEP 0 :PAUSE 5 :NEXT
6120 '
6130 RETURN
6140 '
6980 ' ■■■■■■■■■■■■■■■■
6990 '
7000   LABEL "ｵｰﾌﾟﾆﾝｸﾞ"
7010 '
7020 ' ■■■■■■■■■■■■■■■■
7030 '
7040 FOR I=1 TO 184
7050   COLOR (I MOD 6)+1 :PRINT "TRON ";
7060 NEXT
7080 RETURN
7090 '
7100 '
7980 ' ■■■■■■■■■■■■■■■■■■■
7990 '
8000   LABEL "ｷｬﾗｸﾀｰ･ﾃｲｷﾞ"
8010 '
8020 ' ■■■■■■■■■■■■■■■■■■■
8030 '
8040 DEFCHR$(100)=HEXCHR$("0018183C3C181818667E7E3C3C7E7E66666666000066667E")
8050 DEFCHR$(110)=HEXCHR$("1818183C3C181800667E7E3C3C7E7E667E666600006666666")
8060 DEFCHR$(120)=HEXCHR$("000E7F1818000000000E7FFFFFE70000000000E7E7E70000")
8070 DEFCHR$(130)=HEXCHR$("0070FE18180000000070FEFFFFE7000000000E7E7E70000")
8080 '
8090 DEFCHR$(200)=HEXCHR$("0018183C3C18181866666666000066667E000000000000000018")
8100 DEFCHR$(210)=HEXCHR$("1818183C3C1818007E666600006666661800000000000000")
8110 DEFCHR$(220)=HEXCHR$("000E7F181800000000000E7E7E7000000000000000000000")
8120 DEFCHR$(230)=HEXCHR$("0070FE181800000000000E7E7E7000000000000000000000")
8130 '
8140 DEFCHR$(&H90)=HEXCHR$("000000FFFF000000000000FFFF000000000000FFFF000000")
8150 DEFCHR$(&H91)=HEXCHR$("181818181818181818181818181818181818181818181818")
8160 DEFCHR$(&H97)=HEXCHR$("000000F8F8181818000000F8F8181818000000F8F8181818")
8170 DEFCHR$(&H98)=HEXCHR$("181818F8F8000000181818F8F8000000181818F8F8000000")
8180 DEFCHR$(&H99)=HEXCHR$("1818181F1F0000001818181F1F0000001818181F1F000000")
8190 DEFCHR$(&H9A)=HEXCHR$("0000001F1F1818180000001F1F1818180000001F1F181818")
8200 '
8210 RETURN
8220 '
29970 '
29980 ' ■■■■■   ﾏｼﾝｺﾞ ﾃﾞｰﾀ   ■■■■■
29990 '
30000 DATA 3E,01,32,0D,E0,CD,EC,D0
30010 DATA 3A,06,E0,FE,04,DA,44,D0
30020 DATA 3A,01,E0,21,03,E0,BE,DC
30030 DATA 1B,D1,DA,F7,D1,3A,01,E0
30040 DATA 21,03,E0,BE,C4,58,D1,DA
30050 DATA F7,D1,3A,00,E0,21,02,E0
30060 DATA BE,DC,91,D1,DA,F7,D1,3A
30070 DATA 00,E0,21,02,E0,BE,C4,C4
30080 DATA D1,DA,F7,D1,CD,1B,D1,DA
30090 DATA F7,D1,CD,58,D1,DA,F7,D1
30100 DATA CD,91,D1,DA,F7,D1,CD,C4
30110 DATA D1,DA,F7,D1,3A,0C,E0,FE
30120 DATA 08,20,05,CD,2E,D1,18,15
30130 DATA FE,02,20,05,CD,69,D1,18
30140 DATA 0C,FE,04,20,05,CD,9F,D1
30150 DATA 18,03,CD,D2,D1,3E,01,32
30160 DATA 07,E0,C3,F7,D1,3E,00,32
30170 DATA 0D,E0,CD,EC,D0,3A,04,E0
30180 DATA 21,0C,E0,86,FE,0A,20,03
30190 DATA 21,04,E0,7E,FE,08,20,10
30200 DATA CD,1B,D1,38,08,CD,2E,D1
30210 DATA 3E,01,32,07,E0,C3,F7,D1
30220 DATA FE,02,20,10,CD,58,D1,38
30230 DATA 08,CD,69,D1,3E,01,32,07
30240 DATA E0,C3,F7,D1,FE,04,20,10
30250 DATA CD,91,D1,38,08,CD,9F,D1
30260 DATA 3E,01,32,07,E0,C3,F7,D1
30270 DATA FE,06,20,BC,CD,C4,D1,38
30280 DATA 08,CD,D2,D1,3E,01,32,07
30290 DATA E0,C3,F7,D1,3A,0D,E0,FE
30300 DATA 00,20,0C,7E,32,0C,E0,21
30310 DATA 00,E0,56,23,5E,18,0C,3A
30320 DATA 05,E0,32,0C,E0,21,02,E0
30330 DATA 56,23,5E,21,00,30,4A,16
30340 DATA 00,06,28,19,10,FD,09,22
30350 DATA 08,E0,C9,2A,08,E0,11,28
30360 DATA 00,B7,ED,52,44,4D,ED,78
30370 DATA FE,20,28,0B,B7,C9,2A,08
```

```
30380 DATA E0,11,28,00,B7,ED,52,22
30390 DATA 0A,E0,3E,08,32,0C,E0,3A
30400 DATA 0D,E0,FE,00,20,09,3A,01
30410 DATA E0,3D,32,01,E0,18,07,3A
30420 DATA 03,E0,3D,32,03,E0,37,C9
30430 DATA 2A,08,E0,11,28,00,19,44
30440 DATA 4D,ED,78,FE,20,28,09,B7
30450 DATA C9,2A,08,E0,11,28,00,19
30460 DATA 22,0A,E0,3E,02,32,0C,E0
30470 DATA 3A,0D,E0,FE,00,20,09,3A
30480 DATA 01,E0,3C,32,01,E0,18,07
30490 DATA 3A,03,E0,3C,32,03,E0,37
30500 DATA C9,2A,08,E0,2B,44,4D,ED
30510 DATA 78,FE,20,28,06,B7,C9,2A
30520 DATA 08,E0,2B,22,0A,E0,3E,04
30530 DATA 32,0C,E0,3A,0D,E0,FE,00
30540 DATA 20,09,3A,00,E0,3D,32,00
30550 DATA E0,18,07,3A,02,E0,3D,32
30560 DATA 02,E0,37,C9,2A,08,E0,23
30570 DATA 44,4D,ED,78,FE,20,28,06
30580 DATA B7,C9,2A,08,E0,23,22,0A
30590 DATA E0,3E,06,32,0C,E0,3A,0D
30600 DATA E0,FE,00,20,09,3A,00,E0
30610 DATA 3C,32,00,E0,18,07,3A,02
30620 DATA E0,3C,32,02,E0,37,C9,CD
30630 DATA 26,D2,ED,4B,08,E0,16,00
30640 DATA 3A,0E,E0,5F,CD,3C,D3,3A
30650 DATA 07,E0,FE,00,20,0D,ED,4B
30660 DATA 0A,E0,16,01,3A,0F,E0,5F
30670 DATA C3,3C,D3,ED,4B,0A,E0,16
30680 DATA 02,1E,E8,C3,3C,D3,3A,0C
30690 DATA E0,FE,02,CA,79,D2,FE,04
30700 DATA CA,BA,D2,FE,06,CA,FB,D2
30710 DATA 3A,0D,E0,FE,00,20,05,3A
30720 DATA 04,E0,18,03,3A,05,E0,FE
30730 DATA 04,20,04,3E,99,18,0A,FE
30740 DATA 06,20,04,3E,98,18,02,3E
30750 DATA 91,32,0E,E0,3A,0D,E0,FE
30760 DATA 00,20,0B,3E,64,32,0F,E0
30770 DATA 3E,08,32,04,E0,C9,3E,C8
30780 DATA 32,0F,E0,3E,08,32,05,E0
30790 DATA C9,3A,0D,E0,FE,00,20,05
30800 DATA 3A,04,E0,18,03,3A,05,E0
30810 DATA FE,04,20,04,3E,9A,18,0A
30820 DATA FE,06,20,04,3E,97,18,02
30830 DATA 3E,91,32,0E,E0,3A,0D,E0
30840 DATA FE,00,20,0B,3E,6E,32,0F
30850 DATA E0,3E,02,32,04,E0,C9,3E
30860 DATA D2,32,0F,E0,3E,02,32,05
30870 DATA E0,C9,3A,0D,E0,FE,00,20
30880 DATA 05,3A,04,E0,18,03,3A,05
30890 DATA E0,FE,08,20,04,3E,97,18
30900 DATA 0A,FE,02,20,04,3E,98,18
30910 DATA 02,3E,90,32,0E,E0,3A,0D
30920 DATA E0,FE,00,20,0B,3E,78,32
30930 DATA 0F,E0,3E,04,32,04,E0,C9
30940 DATA 3E,DC,32,0F,E0,3E,04,32
30950 DATA 05,E0,C9,3A,0D,E0,FE,00
30960 DATA 20,05,3A,04,E0,18,03,3A
30970 DATA 05,E0,FE,08,20,04,3E,9A
30980 DATA 18,0A,FE,02,20,04,3E,99
30990 DATA 18,02,3E,90,32,0E,E0,3A
31000 DATA 0D,E0,FE,00,20,0B,3E,82
31010 DATA 32,0F,E0,3E,06,32,04,E0
31020 DATA C9,3E,E6,32,0F,E0,3E,06
31030 DATA 32,05,E0,C9,7A,FE,01,20
31040 DATA 04,3E,27,18,1F,3A,0D,E0
31050 DATA FE,00,20,0D,7A,FE,00,20
31060 DATA 04,3E,26,18,0F,3E,06,18
31070 DATA 0B,7A,FE,00,20,04,3E,22
31080 DATA 18,02,3E,02,CB,A0,ED,79
31090 DATA CB,E0,7B,ED,79,C9,01,00
31100 DATA 20,ED,78,CB,DF,ED,79,03
31110 DATA 78,FE,23,38,F4,79,FE,E8
31120 DATA 38,EF,C9
31130 '
59990 '
60000 ' ****************************************
60010 ' *                                      *
60020 ' *   Sample Game for  X1 ﾏｼﾝｺﾞ ﾆｭｳﾓﾝ    *
60030 ' *                                      *
60040 ' *          BASIC & ﾏｼﾝｺﾞ  version      *
60050 ' *                                      *
60060 ' ****************************************
```

４−25／DISK BASICをお使いの方へ

Ｘ１の周辺機器として、フロッピーディスクを購入し、ＤＩＳＫ　ＢＡＳＩＣ　を使われている方、あるいは新製品Ｘ１Ｄで３インチディスクを使われている方には一言御注意申し上げておくことがあります。

ＤＩＳＫ　ＢＡＳＩＣ　で、ＴＲＯＮのマシン語版を実行すると、

Ｉｌｌｅｇａｌ　ｆｕｎｃｔｉｏｎ　ｃａｌｌ

エラーを生じることがあります。これは、プログラム冒頭の１３０行の

ＣＬＥＡＲ　＆ＨＤ０００

が原因です。ＤＩＳＫ　ＢＡＳＩＣ　では、フリーエリアが減少するために、プログラムの格納場所が不足するのです。このような時は、以下のようにして下さい。

解決法１（安易な方法）

１３０行の　ＣＬＥＡＲ　文を

ＣＬＥＡＲ　＆ＨＦＦ００

に変更します。すると、フリーエリアが広くなってプログラムは一応動くはずです。「一応」と述べたのは、この方法が邪道だからです。本文でも強調したように、このようにすると「マシン語部分」が、上からはＢＡＳＩＣのテキストに迫られ、下からはスタックに迫られるサンドイッチ領域に置かれてしまうからです。もし、これでエラーを生じた時は次の方法で試みて下さい。

解決法２（プログラム分割法）

ＤＡＴＡ文として長々と入っている「マシン語部分」がメモリー不足の原因なので、マシン語部は別にディスクにセーブしておきます。しかる後に、２９９７０行以降のデータ部を

消去します。同時に、１９０行〜２４０行の「マシン語書き込みルーチン」も消去します（２５０行の　USR　関数の定義は消してなりません）。１３０行の　CLEAR文は、

　　CLEAR　&HD000

のまま残します。このようにして、実行時はマシン語をロード、続いて　BASIC　プログラムをロードという２段階で行なえば、TRONゲームは正常に動くはずです。

４－26／最後のコーヒータイム

「トロンゲーム　マシン語版」は期待通り動きましたか？　あのモタモタしていた「オールＢＡＳＩＣ版」が生き返ったように、スピーディでスリリングなゲームになったと思います。この種のリアルタイムゲームの面白さにとって、スピードがいかに大切であるかを再認識されたことでしょう。

これ位ゲームが速くなると、テンキーをガチャガチャ操作して壊してしまうのでは？　と不安になる程ですね。この時には、ジョイスティックを接続して遊ぶとよいでしょう。プログラム中２１３０行に　Ｉ＝ＳＴＩＣＫ（０）　とありますが、これを　Ｉ＝ＳＴＩＣＫ（１）　とすればジョイスティック１に、　Ｉ＝ＳＴＩＣＫ（２）　とすればジョイスティック２に変えることができます。マニュアルの１５４ページを参照して下さい。

以上をもちまして、読者の皆様との共同作業を終わらせていただきます。長い間、おつきあいいただき感謝いたします。

実は、このゲームは私のマシン語ゲーム第１作目でありまして、何かと不備な点もあろうかと思います。皆様の創意工夫で改良されて、Ｘ１のマシン語をマスターする出発点としていただければ、筆者として最高の喜びです。

私たちは本書を通じて、マシン語によりＩ／Ｏポートをアクセスし、代表的な出力装置であるディスプレイテレビの制御法（テキスト画面に限りましたが）をマスターいたしました。また、一つのまとまったマシン語プログラム（ゲーム）を作成することにも成功いたしました。私たちは、マシン語学習における最初の大きな壁を乗り越えたのです！

パソコンテレビＸ１は、すばらしいハードウェアで、マシン語で制御してみたい部分はまだまだ多く残っています。しかし、これらすべてに解説を加えるには、本書のページ数は余りに少なく、他日の機会を待つことにしたいと思います。

私がＸ１のマシン語を研究し始めた頃は、余りにもデータが少なく大変な苦労をいたしましたが、本書が出版される頃には、きっとマイコン雑誌にもＸ１のマシン語の記事が出るようになっていると思います。本書は、読者の皆様がＸ１のマシン語をゼロから出発して、基本的な所までマスターできるように考えて執筆されました。雑誌等の記事や、ＢＡＳＩＣインタプリターの解読を通じて、一歩一歩次の段階めざして勉強を続けて下さい。

きっと、マシン語の素晴らしさ、さらに言えば、２０世紀後半に人類がなしとげた最大の発明の１つと言われている　マイクロコンピューター（マイクロプロセッサー）の素晴らしさを身をもって実感されることでしょう。では、またお会いできる時まで……。

あとがき

■あとがき

本書はパソコンテレビＸ１（ＣＺ８００Ｃ，ＣＺ８００Ｄ）をＺ８０マシン語で制御するためのユーザーの立場で書かれた入門書です。最近Ｘ１シリーズの新製品Ｘ１Ｃ，Ｘ１Ｄが発売されました。これらを含めＸ１シリーズは、いずれもすばらしいハードウェアを持っています。読者の皆様も本書を参考に、Ｘ１の機能を１００％引き出す方法を考えてみて下さい。Ｘ１のハードウェアはきっと皆様の期待に十分応えてくれることでしょう。

本書を書く際に参考にしたＺ８０マシン語の本を挙げます。これらの本には本書で書ききれなかったテーマも書かれていますから、本書を参考に「Ｘ１用」にプログラムを移植してみると力がつくでしょう。

図解マイクロコンピューター　Ｚ－８０の使い方（横田英一著　オーム社刊）

この本はＺ８０ＣＰＵの標準的教科書という性格を持っています。

Ｚ８０マイコンプログラムテクニック（庄司渉，本田稔著　電波新聞社刊）

各命令が丁寧に解説され、とくにフラグ変化についても細かい説明がなされています。

「マイコンピューター　１９８２年第７号」　入門•研究特集　Ｚ８０アセンブラ言語入門（ＣＱ出版社刊）

入出力命令について普通の本には書かれていない詳しい説明があります。私はこの本により、Ｘ１のＩ/０ポートの謎を理解できました。また、割り込みについても貴重なプログラム例が出ています。

ＰＣ－８００１•８８０１　マシン語入門（塚越一雄著　電波新聞社刊）

主にＰＣ－８００１用に書かれていますが、Ｚ８０マシン語学習入門書としても使い得る名著。私のマシン語の勉強は、この本との出会いからスタートしました。

ＰＣ－８００１　マシン語入門Ⅱ（塚越一雄著　電波新聞社刊）

上に挙げた本の続編。アセンブラ指示命令の使い方、ＵＳＲ関数についての説明が出ていま

す。是非「X１用」に翻訳して考えられることをお勧めします。

PC－６００１　マシン語入門（桜田幸嗣・田村明史共著　アスキー出版局刊）

乗除算プログラムやソーティングのテクニックなどはX１においても参考になります。

次には、BASICプログラムも含めX１全般について読者の参考となるであろう本・雑誌記事を挙げます。

SHARPパソコンテレビX１　HuBASIC私の勉強ノート（品川ゆり・Dr．Bee著　ラジオ技術社刊）

X１について最も早く出た本。私は、BASICの学習をしていた初期において、とくにサンプルプログラムの解読でプログラミングの力をつけることができました。

パソコンテレビX１　BASIC（戸川隼人著　サイエンス社刊）

マニュアルだけではよくわからない命令の使い方についても丁寧に説明されています。

パソコンテレビライフ（別冊太陽　リビング　平凡社刊）

X１を家庭で使うためのアイデアに富んだ本。カラー写真が大変美しい。

X１テクニカルマスター（ストラットフォードC．C．C．著　日本ソフトバンク刊）

X１の持つ楽しい機能について易しく説明されています。終わりの方で少しマシン語にも触れていて、グラフィック画面を制御するプログラムが出ています。本書と併読するとよいでしょう。

「ASCII　1983年8月号」LOAD　TEST　SHARP　X１（アスキー編集部）

ハードウェア，ソフトウェアの詳細が出ていて参考になります。とくにサブCPUとの交信法はこの記事でわかりました。

「Ｉ/Ｏ　１９８３年６月号」　Ｘ１全回路図（Ｉ/Ｏ編集部）

メーカー未発表の回路図が編集部の責任において掲載されており、ユーザーにとっては貴重な資料です。

「Ｉ/Ｏ　１９８３年３月号」　Ｘ１用ミニアセンブラ（ｄＢ　ＳＯＦＴ）

Ｘ１については、著者の知る限り唯一発表されているアセンブラです。小規模のマシン語プログラム作りには威力を発揮します。本書でも一部利用しました。付録５がそのソースリストです。

「マイコン　１９８３年１月号」　パソコンの互換性を探る（高橋雄一）

Ｘ１　ＨｕＢＡＳＩＣの中間コード表が出ています。

「マイコン　１９８３年２月号」　パソコンテレビＸ１とＭＺ－７００の互換性（高橋雄一）

ＢＡＳＩＣインタプリターの有効利用にとって不可欠なＨｕＢＡＳＩＣの内部解析結果が出ています。この資料を指針としてＢＡＳＩＣを解読すると、時間を節約することができます。

「Ｏｈ！ＭＺ　１９８３年５，６月号」　Ｘ１内部サブルーチンの解析（イッティ・リッターポーン）

ＢＡＳＩＣインタプリターの初めの方にあるサブルーチンの使い方が述べられています。

「Ｏｈ！ＭＺ　１９８３年７月号」　Ｘ１逆アセンブラと命令語の解析（イッティ・リッターポーン）

「マイコン　１９８３年７月号」　Ｘ１ディスアセンブラ（高橋雄一）

上記２つの記事はいずれも、「マシン語→ニーモニック」という逆変換を行なうプログラム（逆アセンブラという）を掲載しています。逆アセンブラはＢＡＳＩＣの内部ルーチンの解読には欠かせません。

以上はごく一部です。この他にも多くの雑誌にＸ１のプログラム（大半がＢＡＳＩＣですが）が発表されていますから、参考にして下さい。

最後になりましたが、コンピューターに関して全く無名の私の企画を採用して、本にまとめ上げて下さった日本ソフト＆ハード社編集部の皆様ならびにコンピュータイレブン横浜駅西口店の皆様に感謝いたします。

１９８３年１０月

著者記す。

付録1　Ｚ８０　命令表

用いる記号

○ n　は　　８ビット定数

○ nn′は　　１６ビット定数，nが上位８ビット，n′が下位８ビット

（ニーモニックではラベル名を書いてもよい。）

○ d　は　　−128〜+127　の範囲（符号付１バイト１６進数）のディスプレイスメント。

○ e　は　　−128〜+127　の範囲（符号付１バイト16進数）の相対アドレス。

次の命令の先頭アドレスを０とする。

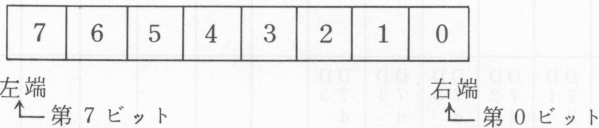
（ニーモニックではラベル名や絶対アドレスを書いてもよい。ただし，アセンブラにより異なる場合もある。）

○ Cy は　　キャリーフラグ

○ 添字Ｈは上位８ビット（high），添字Ｌは下位８ビット（low）を意味する。

○ ビット操作命令　SET，RES，BIT で，ビット番号は下記のとおりである。

7	6	5	4	3	2	1	0

左端　↑第７ビット　　　　右端　↑第０ビット

○ Ｘ１では I/0 ポートが 64 Ｋバイトあるので，特に次の記号を用いる。

I/0（BC）は，BC レジスタペアの内容番地のポートを表わす。

I/0（An）は，Ａレジスタを上位８ビット，定数ｎを下位８ビットとする番地のポートを表わす。

○ フラグ変化

×　　不定

1　　１になる（セットされる）

0　　０になる（リセットされる）

•　　状態にしたがって，セット・リセットされる

−　　変化せず

8ビットロード命令

×	A	B	C	D	E	H	L	(HL)	(BC)	(DE)	(IX+d)	(IY+d)	(nn′)	n	I	R
LD A, ×	7F	78	79	7A	7B	7C	7D	7E	0A	1A	DD 7E d	FD 7E d	3A n′ n	3E n	ED 57	ED 5F
LD B, ×	47	40	41	42	43	44	45	46			DD 46 d	FD 46 d		06 n		
LD C, ×	4F	48	49	4A	4B	4C	4D	4E			DD 4E d	FD 4E d		0E n		
LD D, ×	57	50	51	52	53	54	55	56			DD 56 d	FD 56 d		16 n		
LD E, ×	5F	58	59	5A	5B	5C	5D	5E			DD 5E d	FD 5E d		1E n		
LD H, ×	67	60	61	62	63	64	65	66			DD 66 d	FD 66 d		26 n		
LD L, ×	6F	68	69	6A	6B	6C	6D	6E			DD 6E d	FD 6E d		2E n		
LD (HL), ×	77	70	71	72	73	74	75							36 n		
LD (BC), ×	02															
LD (DE), ×	12															
LD (IX+d), ×	DD 77 d	DD 70 d	DD 71 d	DD 72 d	DD 73 d	DD 74 d	DD 75 d							DD 36 d n		
LD (IY+d), ×	FD 77 d	FD 70 d	FD 71 d	FD 72 d	FD 73 d	FD 74 d	FD 75 d							FD 36 d n		
LD (nn′), ×	32 n′ n															
LD I, ×	ED 47															
LD R, ×	ED 4F															

16ビットロード命令

×	AF	BC	DE	HL	SP	IX	IY	nn′	(nn′)
LD AF, ×									
LD BC, ×								01 n′ n	ED 4B n′ n
LD DE, ×								11 n′ n	ED 5B n′ n
LD HL, ×								21 n′ n	2A n′ n
LD SP, ×				F9		DD F9	FD F9	31 n′ n	ED 7B n′ n
LD IX, ×								DD 21 n′ n	DD 2A n′ n
LD IY, ×								FD 21 n′ n	FD 2A n′ n
LD (nn′), ×		ED 43 n′ n	ED 53 n′ n	22 n′ n	ED 73 n′ n	DD 22 n′ n	FD 22 n′ n		
PUSH ×	F5	C5	D5	E5		DD E5	FD E5		
POP ×	F1	C1	D1	E1		DD E1	FD E1		

交換・ブロック命令

交　換

EX AF, AF	08
EX DE, HL	EB
EX (SP), HL	E3
EX (SP), IX	DD E3
EX (SP), IY	FD E3
EXX	D9

ブロック転送

LDI	ED A0
LDIR	ED B0
LDD	ED A8
LDDR	ED B8

ブロックサーチ

CPI	ED A1
CPIR	ED B1
CPD	ED A9
CPDR	ED B9

8ビット算術・論理演算命令

×	A	B	C	D	E	H	L	(HL)	(IX+d)	(IY+d)	n
ADD A, ×	8 7	8 0	8 1	8 2	8 3	8 4	8 5	8 6	D D 8 6 d	F D 8 6 d	C 6 n
ADC A, ×	8 F	8 8	8 9	8 A	8 B	8 C	8 D	8 E	D D 8 E d	F D 8 E d	C E n
SUB ×	9 7	9 0	9 1	9 2	9 3	9 4	9 5	9 6	D D 9 6 d	F D 9 6 d	D 6 n
SBC A, ×	9 F	9 8	9 9	9 A	9 B	9 C	9 D	9 E	D D 9 E d	F D 9 E d	D E n
AND ×	A 7	A 0	A 1	A 2	A 3	A 4	A 5	A 6	D D A 6 d	F D A 6 d	E 6 n
XOR ×	A F	A 8	A 9	A A	A B	A C	A D	A E	D D A E d	F D A E d	E E n
OR ×	B 7	B 0	B 1	B 2	B 3	B 4	B 5	B 6	D D B 6 d	F D B 6 d	F 6 n
CP ×	B F	B 8	B 9	B A	B B	B C	B D	B E	D D B E d	F D B E d	F E n
INC ×	3 C	0 4	0 C	1 4	1 C	2 4	2 C	3 4	D D 3 4 d	F D 3 4 d	
DEC ×	3 D	0 5	0 D	1 5	1 D	2 5	2 D	3 5	D D 3 5 d	F D 3 5 d	

16ビット算術演算命令

×	B C	D E	H L	S P	I X	I Y
ADD HL, ×	0 9	1 9	2 9	3 9		
ADD IX, ×	D D 0 9	D D 1 9		D D 3 9	D D 2 9	
ADD IY, ×	F D 0 9	F D 1 9		F D 3 9		F D 2 9
ADC HL, ×	E D 4 A	E D 5 A	E D 6 A	E D 7 A		
SBC HL, ×	E D 4 2	E D 5 2	E D 6 2	E D 7 2		
INC ×	0 3	1 3	2 3	3 3	D D 2 3	F D 2 3
DEC ×	0 B	1 B	2 B	3 B	D D 2 B	F D 2 B

ビット操作命令

×	A	B	C	D	E	H	L	(HL)	(IX+d)	(IY+d)
BIT 0, ×	CB 47	CB 40	CB 41	CB 42	CB 43	CB 44	CB 45	CB 46	DD CB d 46	FD CB d 46
BIT 1, ×	CB 4F	CB 48	CB 49	CB 4A	CB 4B	CB 4C	CB 4D	CB 4E	DD CB d 4E	FD CB d 4E
BIT 2, ×	CB 57	CB 50	CB 51	CB 52	CB 53	CB 54	CB 55	CB 56	DD CB d 56	FD CB d 56
BIT 3, ×	CB 5F	CB 58	CB 59	CB 5A	CB 5B	CB 5C	CB 5D	CB 5E	DD CB d 5E	FD CB d 5E
BIT 4, ×	CB 67	CB 60	CB 61	CB 62	CB 63	CB 64	CB 65	CB 66	DD CB d 66	FD CB d 66
BIT 5, ×	CB 6F	CB 68	CB 69	CB 6A	CB 6B	CB 6C	CB 6D	CB 6E	DD CB d 6E	FD CB d 6E
BIT 6, ×	CB 77	CB 70	CB 71	CB 72	CB 73	CB 74	CB 75	CB 76	DD CB d 76	FD CB d 76
BIT 7, ×	CB 7F	CB 78	CB 79	CB 7A	CB 7B	CB 7C	CB 7D	CB 7E	DD CB d 7E	FD CB d 7E
RES 0, ×	CB 87	CB 80	CB 81	CB 82	CB 83	CB 84	CB 85	CB 86	DD CB d 86	FD CB d 86
RES 1, ×	CB 8F	CB 88	CB 89	CB 8A	CB 8B	CB 8C	CB 8D	CB 8E	DD CB d 8E	FD CB d 8E
RES 2, ×	CB 97	CB 90	CB 91	CB 92	CB 93	CB 94	CB 95	CB 96	DD CB d 96	FD CB d 96
RES 3, ×	CB 9F	CB 98	CB 99	CB 9A	CB 9B	CB 9C	CB 9D	CB 9E	DD CB d 9E	FD CB d 9E
RES 4, ×	CB A7	CB A0	CB A1	CB A2	CB A3	CB A4	CB A5	CB A6	DD CB d A6	FD CB d A6
RES 5, ×	CB AF	CB A8	CB A9	CB AA	CB AB	CB AC	CB AD	CB AE	DD CB d AE	FD CB d AE
RES 6, ×	CB B7	CB B0	CB B1	CB B2	CB B3	CB B4	CB B5	CB B6	DD CB d B6	FD CB d B6
RES 7, ×	CB BF	CB B8	CB B9	CB BA	CB BB	CB BC	CB BD	CB BE	DD CB d BE	FD CB d BE
SET 0, ×	CB C7	CB C0	CB C1	CB C2	CB C3	CB C4	CB C5	CB C6	DD CB d C6	FD CB d C6
SET 1, ×	CB CF	CB C8	CB C9	CB CA	CB CB	CB CC	CB CD	CB CE	DD CB d CE	FD CB d CE
SET 2, ×	CB D7	CB D0	CB D1	CB D2	CB D3	CB D4	CB D5	CB D6	DD CB d D6	FD CB d D6
SET 3, ×	CB DF	CB D8	CB D9	CB DA	CB DB	CB DC	CB DD	CB DE	DD CB d DE	FD CB d DE
SET 4, ×	CB E7	CB E0	CB E1	CB E2	CB E3	CB E4	CB E5	CB E6	DD CB d E6	FD CB d E6
SET 5, ×	CB EF	CB E8	CB E9	CB EA	CB EB	CB EC	CB ED	CB EE	DD CB d EE	FD CB d EE
SET 6, ×	CB F7	CB F0	CB F1	CB F2	CB F3	CB F4	CB F5	CB F6	DD CB d F6	FD CB d F6
SET 7, ×	CB FF	CB F8	CB F9	CB FA	CB FB	CB FC	CB FD	CB FE	DD CB d FE	FD CB d FE

アキュムレータ・フラグ・CPU制御命令

アキュムレータ・フラグ制御

DAA	27
CPL	2F
NEG	ED 44
CCF	3F
SCF	37

CPU制御

NOP	00
HALT	76
DI	F3
EI	FB
IM 0	ED 46
IM 1	ED 56
IM 2	ED 5E

ローテート・シフト命令

×	A	B	C	D	E	H	L	(HL)	(IX+d)	(IY+d)
RLC ×	CB 07	CB 00	CB 01	CB 02	CB 03	CB 04	CB 05	CB 06	DD CB d 06	FD CB d 06
RRC ×	CB 0F	CB 08	CB 09	CB 0A	CB 0B	CB 0C	CB 0D	CB 0E	DD CB d 0E	FD CB d 0E
RL ×	CB 17	CB 10	CB 11	CB 12	CB 13	CB 14	CB 15	CB 16	DD CB d 16	FD CB d 16
RR ×	CB 1F	CB 18	CB 19	CB 1A	CB 1B	CB 1C	CB 1D	CB 1E	DD CB d 1E	FD CB d 1E
SLA ×	CB 27	CB 20	CB 21	CB 22	CB 23	CB 24	CB 25	CB 26	DD CB d 26	DD CB d 26
SRA ×	CB 2F	CB 28	CB 29	CB 2A	CB 2B	CB 2C	CB 2D	CB 2E	DD CB d 2E	FD CB d 2E
SRL ×	CB 3F	CB 38	CB 39	CB 3A	CB 3B	CB 3C	CB 3D	CB 3E	DD CB d 3E	FD CB d 3E
RLD								ED 6F		
RRD								ED 67		

RLCA	07
RRCA	0F
RLA	17
RRA	1F

ジャンプ命令

×	無条件	キャリー C	ノン キャリー NC	ゼロ Z	ノン ゼロ NZ	パリティ偶数 PE	パリティ奇数 PO	負 M	正 P	カウント
JP ×, nn′	C3 n′ n	DA n′ n	D2 n′ n	CA n′ n	C2 n′ n	EA n′ n	E2 n′ n	FA n′ n	F2 n′ n	
JR ×, e	18 e	38 e	30 e	28 e	20 e					
JP (HL)	E9									
JP (IX)	DD E9									
JP (IY)	FD E9									
DJNZ e										10 e

コール・リターン命令

コール・リターン

×	無条件	キャリー C	ノン キャリー NC	ゼロ Z	ノン ゼロ NZ	パリティ偶数 PE	パリティ奇数 PO	負 M	正 P
CALL ×, nn′	CD n′ n	DC n′ n	D4 n′ n	CC n′ n	C4 n′ n	EC n′ n	E4 n′ n	FC n′ n	F4 n′ n
RET ×	C9	D8	D0	C8	C0	E8	E0	F8	F0
RETI	ED 4D								
RETN	ED 45								

リスタート

×	00H	08H	10H	18H	20H	28H	30H	38H
RST ×	C7	CF	D7	DF	E7	EF	F7	FF

入出力命令

入力

×＼	A	B	C	D	E	H	L
IN ×, (n)	DB n						
IN ×, (C)	ED 78	ED 40	ED 48	ED 50	ED 58	ED 60	ED 68

ブロック入力コマンド

INI	ED A2
INIR	ED B2
IND	ED AA
INDR	ED BA

出力

×＼	A	B	C	D	E	H	L
OUT (n), ×	D3 n						
OUT (C), ×	ED 79	ED 41	ED 49	ED 51	ED 59	ED 61	ED 69

ブロック出力コマンド

OUTI	ED A3
OTIR	ED B3
OUTD	ED AB
OTDR	ED BB

ニーモニック	マシン語	S	Z	H	P	V	N	C	所要クロックサイクル	動作
		フラグ変化			P/V					
ADC A, n	CE n								7	8ビット加算キャリ付
ADC A, A	8F									(アド・ウィズ・キャリ)
ADC A, B	88									A←A＋ソース＋Cy
ADC A, C	89									
ADC A, D	8A								4	
ADC A, E	8B	・	・	・	ー	・	0	・		
ADC A, H	8C									
ADC A, L	8D									
ADC A,(HL)	8E								7	
ADC A,(IX+d)	DD 8E d								19	
ADC A,(IY+d)	FD 8E d								19	
ADC HL, BC	ED 4A									16ビット加算キャリ付
ADC HL, DE	ED 5A									(アド・ウィズ・キャリ)
ADC HL, HL	ED 6A	・	・	×	ー	・	0	・	15	HL←HL＋ソース＋Cy
ADC HL, SP	ED 7A									
ADD A, n	C6 n								7	8ビット加算（アド）
ADD A, A	87									
ADD A, B	80									
ADD A, C	81									
ADD A, D	82								4	A←A＋ソース
ADD A, E	83	・	・	・	ー	・	0	・		
ADD A, H	84									
ADD A, L	85									
ADD A,(HL)	86								7	
ADD A,(IX+d)	DD 86 d								19	
ADD A,(IY+d)	FD 86 d								19	
ADD HL, BC	09									16ビット加算（アド）
ADD HL, DE	19								11	
ADD HL, HL	29									HL←HL＋ソース
ADD HL, SP	39									
ADD IX, BC	DD 09									
ADD IX, DE	DD 19	ー	ー	×	ー	ー	0	・		IX← IX＋ソース
ADD IX, IX	DD 29								15	
ADD IX, SP	DD 39									

ニーモニック	マシン語	S	Z	H	P/V P	P/V V	N	C	所要クロックサイクル	動作
ADD IY, BC	FD 09									IY←IY＋ソース
ADD IY, DE	FD 19									
ADD IY, IY	FD 29									
ADD IY, SP	FD 39									
AND n	E6 n								7	論理積（アンド）
AND A	A7									
AND B	A0									A←A∧ソース
AND C	A1									
AND D	A2								4	
AND E	A3	・	・	1	・	—	0	0		
AND H	A4									
AND L	A5									
AND (HL)	A6								7	
AND (IX+d)	DD A6 d								19	
AND (IY+d)	FD A6 d								19	
BIT 0, A	CB 47									ビットテスト
BIT 0, B	CB 40									
BIT 0, C	CB 41									
BIT 0, D	CB 42								8	ソースの第0ビットを調べZフラグを設定
BIT 0, E	CB 43	×	・	1	×	—	0	—		
BIT 0, H	CB 44									
BIT 0, L	CB 45									
BIT 0,(HL)	CB 46								12	
BIT 0,(IX+d)	DD CB d 46								20	
BIT 0,(IY+d)	FD CB d 46								20	
BIT 1, A	CB 4F									ビットテスト
BIT 1, B	CB 48									
BIT 1, C	CB 49									
BIT 1, D	CB 4A								8	ソースの第1ビットを調べZフラグを設定
BIT 1, E	CB 4B	×	・	1	×	—	0	—		
BIT 1, H	CB 4C									
BIT 1, L	CB 4D									
BIT 1,(HL)	CB 4E								12	
BIT 1,(IX+d)	DD CB d 4E								20	
BIT 1,(IY+d)	FD CB d 4E								20	

a	b	答
0	0	0
0	1	0
1	0	0
1	1	1

ニーモニック	マシン語	フラグ変化 S	Z	H	P/V P	V	N	C	所要クロックサイクル	動作
BIT 2, A	CB 57									ビットテスト
BIT 2, B	CB 50									
BIT 2, C	CB 51									
BIT 2, D	CB 52								8	ソースの第2ビットを調べ
BIT 2, E	CB 53									Zフラグを設定
BIT 2, H	CB 54	×	•	1	×	−	0	−		
BIT 2, L	CB 55									
BIT 2,(HL)	CB 56								12	
BIT 2,(IX+d)	DD CB d 56								20	
BIT 2,(IY+d)	FD CB d 56								20	
BIT 3, A	CB 5F									ビットテスト
BIT 3, B	CB 58									
BIT 3, C	CB 59									
BIT 3, D	CB 5A								8	ソースの第3ビットを調べ
BIT 3, E	CB 5B									Zフラグを設定
BIT 3, H	CB 5C	×	•	1	×	−	0	−		
BIT 3, L	CB 5D									
BIT 3,(HL)	CB 5E								12	
BIT 3,(IX+d)	DD CB d 5E								20	
BIT 3,(IY+d)	FD CB d 5E								20	
BIT 4, A	CB 67									ビットテスト
BIT 4, B	CB 60									
BIT 4, C	CB 61									
BIT 4, D	CB 62								8	ソースの第4ビットを調べ
BIT 4, E	CB 63									Zフラグを設定
BIT 4, H	CB 64	×	•	1	×	−	0	−		
BIT 4, L	CB 65									
BIT 4,(HL)	CB 66								12	
BIT 4,(IX+d)	DD CB d 66								20	
BIT 4,(IY+d)	FD CB d 66								20	
BIT 5, A	CB 6F									ビットテスト
BIT 5, B	CB 68									
BIT 5, C	CB 69									
BIT 5, D	CB 6A								8	ソースの第5ビットを調べ
BIT 5, E	CB 6B	×	•	1	×	−	0	−		Zフラグを設定

ニーモニック	マシン語	フラグ変化 S	Z	H	P/V P	P/V V	N	C	所要クロックサイクル	動作
BIT 5, H	CB 6C									
BIT 5, L	CB 6D									
BIT 5, (HL)	CB 6E								12	
BIT 5, (IX+d)	DD CB d 6E								20	
BIT 5, (IY+d)	FD CB d 6E								20	
BIT 6, A	CB 77									ビットテスト
BIT 6, B	CB 70									
BIT 6, C	CB 71									
BIT 6, D	CB 72								8	ソースの第6ビットを調べ
BIT 6, E	CB 73	×	・	1	×	—	0	—		Zフラグを設定
BIT 6, H	CB 74									
BIT 6, L	CB 75									
BIT 6, (HL)	CB 76								12	
BIT 6, (IX+d)	DD CB d 76								20	
BIT 6, (IY+d)	FD CB d 76								20	
BIT 7, A	CB 7F									ビットテスト
BIT 7, B	CB 78									
BIT 7, C	CB 79									
BIT 7, D	CB 7A								8	ソースの第7ビットを調べ
BIT 7, E	CB 7B	×	・	1	×	—	0	—		Zフラグを設定
BIT 7, H	CB 7C									
BIT 7, L	CB 7D									
BIT 7, (HL)	CB 7E								12	
BIT 7, (IX+d)	DD CB d 7E								20	
BIT 7, (IY+d)	FD CB d 7E								20	
CALL NZ, nn′	C4 n′ n									
CALL Z, nn′	CC n′ n								成立時	サブルーチン・コール（条件付）
CALL NC, nn′	D4 n′ n								17	・条件が成立すれば戻り番地〔PC〕をスタックへPUSHしnn′へジャンプ〔PC← nn′〕
CALL C, nn′	DC n′ n	—	—	—	—	—	—	—	不成立	
CALL PO, nn′	E4 n′ n								10	
CALL PE, nn′	EC n′ n									・成立しなければ本命令は無視する
CALL P, nn′	F4 n′ n									
CALL M, nn′	FC n′ n									
CALL nn′	CD n′ n	—	—	—	—	—	—	—	17	サブルーチン・コール（無条件）PCをスタックへPUSHしPC← nn′

ニーモニック	マシン語	フラグ変化 S	Z	H	P/V P	V	N	C	所要クロックサイクル	動作
CCF	3F	—	—	×	—	—	0	・	4	Cy を反転〔Cy ← $\overline{Cy}$〕
CP n	FE n								7	比較（コンペア） A—ソースの演算をする Aの内容は変わらずフラグだけが変化する
CP A	BF									
CP B	B8									
CP C	B9									
CP D	BA								4	
CP E	BB	・	・	・	—	・	1	・		
CP H	BC									
CP L	BD									
CP (HL)	BE								7	
CP (IX+d)	DD BE d								19	
CP (IY+d)	FD BE d								19	
CPD	ED A9	×	・	×	—	・	1	—	16	比較（コンペア・ディクリメント） A－(HL)のフラグ変化のみ HL←HL－1　BC←BC－1
CPDR	ED B9	×	・	×	—	・	1	—	1バイトにつき 21 最終のみ 16	比較（コンペア・ディクリメント・リピート） CPDをA＝(HL)〔Zフラグ＝1〕またはBC＝0〔Vフラグ＝0〕までくり返す
CPI	ED A1	×	・	×	—	・	1	—	16	比較（コンペア・インクリメント） A－(HL)のフラグ変化のみ HL←HL＋1　BC←BC－1
CPIR	ED B1	×	・	×	—	・	1	—	1バイトにつき 21 最終のみ 16	比較（コンペア・インクリメント・リピート） CPIをA＝(HL)〔Zフラグ＝1〕またはBC＝0〔Vフラグ＝0〕までくり返す
CPL	2F	—	—	1	—	—	1	—	4	コンプリメント Aレジスタに対しビット反転 0→1 1→0
DAA	27	・	・	・	・	—	—	・	4	デジマル・アジャスト・アキュムレーター Aレジスタに対し10進補正

ニーモニック	マシン語	フラグ変化 S	Z	H	P/V P	V	N	C	所要クロックサイクル	動作
DEC A	3D	・	・	・	—	・	1	—	4	8ビットディクリメント ソース←ソース－1
DEC B	05									
DEC C	0D									
DEC D	15									
DEC E	1D									
DEC H	25									
DEC L	2D									
DEC (HL)	35								11	
DEC (IX+d)	DD 35 d								23	
DEC (IY+d)	FD 35 d								23	
DEC BC	0B	—	—	—	—	—	—	—	6	16ビットディクリメント ソース←ソース－1
DEC DE	1B									
DEC HL	2B									
DEC SP	3B									
DEC IX	DD 2B								10	
DEC IY	FD 2B								10	
DI	F3	—	—	—	—	—	—	—	4	割り込み禁止（ディセーブル・インタラプト）
DJNZ e	10 e	—	—	—	—	—	—	—	B＝0 8 B≠0 13	（ディクリメント・ジャンプノンゼロ） B←B－1　B≠0ならeバイトだけジャンプ〔PC←PC＋e〕 B＝0ならジャンプせず〔e＝0と同じ〕
EI	FB	—	—	—	—	—	—	—	4	割り込み許可（インネーブル・インタラプト）
EX (SP), HL	E3	—	—	—	—	—	—	—	19	交換（エクスチェンジ） ソースとディスティネーションの内容を交換する
EX (SP), IX	DD E3								23	
EX (SP), IY	FD E3								23	
EX AF, AF′	08								4	
EX DE, HL	EB								4	
EXX	D9	—	—	—	—	—	—	—	4	BC DE HLの内容と BC′ DE′ HL′の内容を交換する
HALT	76	—	—	—	—	—	—	—	4	命令実行の進行を止めリセットまたは割り込み待ちとなる（ホルト）

ニーモニック	マシン語	フラグ変化 S	Z	H	P/V P	V	N	C	所要クロックサイクル	動作
IM 0	ED 46								8	割り込みモードを0に設定する
IM 1	ED 56	—	—	—	—	—	—	—	8	割り込みモードを1に設定する
IM 2	ED 5E								8	割り込みモードを2に設定する
INC A	3C									8ビットインクリメント
INC B	04									
INC C	0C									
INC D	14								4	ソース←ソース+1
INC E	1C	•	•	•	—	•	0	—		
INC H	24									
INC L	2C									
INC (HL)	34								11	
INC (IX+d)	DD 34 d								23	
INC (IY+d)	FD 34 d								23	
INC BC	03									16ビットインクリメント
INC DE	13								6	
INC HL	23	—	—	—	—	—	—	—		ソース←ソース+1
INC SP	33									
INC IX	DD 23								10	
INC IY	FD 23								10	
IN A,(C)	ED 78									入力
IN B,(C)	ED 40									BCレジスタの内容番地のポー
IN C,(C)	ED 48									トからディスティネーションの
IN D,(C)	ED 50	•	•	0	•	—	0	—	12	レジスタへ入力
IN E,(C)	ED 58									〔ディスティネーション←I/0(BC)〕
IN H,(C)	ED 60									
IN L,(C)	ED 68									
IN A,(n)	DB n	—	—	—	—	—	—	—	11	入力 A←I/0(An)
IND	ED AA	×	•	×	×	—	×	×	16	イン・ディクリメント (HL)←I/0(BC), HL←HL−1 B←B−1

ニーモニック	マシン語	フラグ変化 S	Z	H	P/V P	V	N	C	所要クロックサイクル	動作
INDR	ED BA	×	1	×	×	−	×	×	1バイトにつき 21 最終のみ 16	イン・ディクリメント・リピート INDをB＝0までくり返す
INI	ED A2	×	•	×	×	−	×	×	16	イン・インクリメント (HL)←I/0(BC), HL←HL＋1 B←B－1
INIR	ED B2	×	1	×	×	−	×	×	1バイトにつき 21 最終のみ 16	イン・インクリメント・リピート INIをB＝0までくり返す
JP (HL) JP (IX) JP (IY) JP nn′	E9 DD E9 FD E9 C3 n′ n	−	−	−	−	−	−	−	4 8 8 10	ジャンプ 各レジスタの内容番地へジャンプ 〔PC←HLなど〕 nn′番地へジャンプ〔PC←nn′〕
JP NZ, nn′ JP Z, nn′ JP NC, nn′ JP C, nn′ JP PO, nn′ JP PE, nn′ JP P, nn′ JP M, nn′	C2 n′ n CA n′ n D2 n′ n DA n′ n E2 n′ n EA n′ n F2 n′ n FA n′ n	−	−	−	−	−	−	−	10	ジャンプ（条件付） ・条件が成立すればnn′番地へジャンプ (PC←nn′) ・不成立なら本命令は無視する
JR e	18 e	−	−	−	−	−	−	−	12	ジャンプ・リラティブ（無条件） eバイト先へジャンプする 〔PC←PC＋e〕
JR NZ, e JR Z, e JR NC, e JR C, e	20 e 28 e 30 e 38 e	−	−	−	−	−	−	−	条件成立 12 条件	ジャンプ・リラティブ（条件付） ・条件が成立すればeバイト先へジャンプ 〔PC←PC＋e〕

ニーモニック	マシン語	S	Z	H	P/V P V	N	C	所要クロックサイクル	動作
								不成立 7	・不成立なら本命令は無視する
LD A, n	3E n							7	8ビット転送（ロード）
LD A, A	7F								
LD A, B	78								A←ソース
LD A, C	79								
LD A, D	7A							4	
LD A, E	7B								
LD A, H	7C								
LD A, L	7D	—	—	—	—	—	—		
LD A,(nn′)	3A n′ n							13	
LD A,(BC)	0A							7	
LD A,(DE)	1A							7	
LD A,(HL)	7E							7	
LD A,(IX+d)	DD 7E d							19	
LD A,(IY+d)	FD 7E d							19	
LD A, I	ED 57							9	8ビット転送　A←I
LD A, R	ED 5F							9	A←R
		•	•	0	IFF	0	—		IFF：0のとき割り込み禁止（DI） 1のとき割り込み可（EI） になっている LD A, I LD A, R ではこの値がP/Vにコピーされる
LD B, n	06 n							7	8ビット転送（ロード）
LD B, A	47								
LD B, B	40								B←ソース
LD B, C	41								
LD B, D	42							4	
LD B, E	43	—	—	—	—	—	—		
LD B, H	44								
LD B, L	45								
LD B,(HL)	46							7	
LD B,(IX+d)	DD 46 d							19	
LD B,(IY+d)	FD 46 d							19	

ニーモニック	マシン語	フラグ変化 S	Z	H	P/V P	V	N	C	所要クロックサイクル	動作
LD C, n	0E n								7	8ビット転送（ロード）
LD C, A	4F									
LD C, B	48									C←ソース
LD C, C	49									
LD C, D	4A								4	
LD C, E	4B	—	—	—	—	—	—	—		
LD C, H	4C									
LD C, L	4D									
LD C,(HL)	4E								7	
LD C,(IX+d)	DD 4E d								19	
LD C,(IY+d)	FD 4E d								19	
LD D, n	16 n								7	8ビット転送（ロード）
LD D, A	57									
LD D, B	50									D←ソース
LD D, C	51									
LD D, D	52								4	
LD D, E	53	—	—	—	—	—	—	—		
LD D, H	54									
LD D, L	55									
LD D,(HL)	56								7	
LD D,(IX+d)	DD 56 d								19	
LD D,(IY+d)	FD 56 d								19	
LD E, n	1E n								7	8ビット転送（ロード）
LD E, A	5F									
LD E, B	58									E←ソース
LD E, C	59									
LD E, D	5A								4	
LD E, E	5B	—	—	—	—	—	—	—		
LD E, H	5C									
LD E, L	5D									
LD E,(HL)	5E								7	
LD E,(IX+d)	DD 5E d								19	
LD E,(IY+d)	FD 5E d								19	
LD H, n	26 n								7	8ビット転送（ロード）
LD H, A	67									
LD H, B	60									H←ソース

ニーモニック	マシン語	S	Z	H	P/V	N	C	所要クロックサイクル	動作
LD H, C	61								
LD H, D	62							4	
LD H, E	63	—	—	—	—	—	—		
LD H, H	64								
LD H, L	65								
LD H,(HL)	66							7	
LD H,(IX+d)	DD 66 d							19	
LD H,(IY+d)	FD 66 d							19	
LD L, n	2E n							7	8ビット転送（ロード）
LD L, A	6F								
LD L, B	68								L←ソース
LD L, C	69								
LD L, D	6A							4	
LD L, E	6B	—	—	—	—	—	—		
LD L, H	6C								
LD L, L	6D								
LD L,(HL)	6E							7	
LD L,(IX+d)	DD 6E d							19	
LD L,(IY+d)	FD 6E d							19	
LD I, A	ED 47	—	—	—	—	—	—	9	8ビット転送 I←A
LD R, A	ED 4F							9	R←A
LD (nn′), A	32 n′ n							13	8ビット転送（nn′）←A
LD (BC), A	02	—	—	—	—	—	—	7	（BC）←A
LD (DE), A	12							7	（DE）←A
LD (HL), n	36 n							10	8ビット転送（ロード）
LD (HL), A	77								
LD (HL), B	70								
LD (HL), C	71								（HL）←ソース
LD (HL), D	72	—	—	—	—	—	—	7	
LD (HL), E	73								
LD (HL), H	74								
LD (HL), L	75								

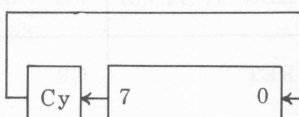
ニーモニック	マシン語	S	Z	H	P	V	N	C	所要クロックサイクル	動作	

LD (IX+d),n
LD (IX+d),A
LD (IX+d),B
LD (IX+d),C
LD (IX+d),D
LD (IX+d),E
LD (IX+d),H
LD (IX+d),L
	DD 36 d n
	DD 77 d
	DD 70 d
	DD 71 d
	DD 72 d
	DD 73 d
	DD 74 d
	DD 75 d
		—	—	—	—	—	—	—	19	8ビット転送（ロード）
										(IX+d) ←ソース	

LD (IY+d),n
LD (IY+d),A
LD (IY+d),B
LD (IY+d),C
LD (IY+d),D
LD (IY+d),E
LD (IY+d),H
LD (IY+d),L
	FD 36 d n
	FD 77 d
	FD 70 d
	FD 71 d
	FD 72 d
	FD 73 d
	FD 74 d
	FD 75 d
		—	—	—	—	—	—	—	19	8ビット転送（ロード）
										(IY+d) ←ソース	

LD BC, nn′ LD BC,(nn′)	01 n′ n ED 4B n′ n	—	—	—	—	—	—	—	10 20	16ビット転送 BC←ソース	nn′ ：定数 (nn′) ：メモリの内容 メモリからレジスタの場合，たとえばHL，(nn′)では L←(nn′) H←(nn′+1) となる
LD DE, nn′ LD DE,(nn′)	11 n′ n ED 5B n′ n	—	—	—	—	—	—	—	10 20	16ビット転送 DE←ソース	
LD HL, nn′ LD HL,(nn′)	21 n′ n 2A n′ n	—	—	—	—	—	—	—	10 16	16ビット転送 HL←ソース	

LD SP, nn′
LD SP,(nn′)
LD SP, HL
LD SP, IX
LD SP, IY
	31 n′ n
	ED 7B n′ n
	F9
	DD F9
	FD F9
		—	—	—	—	—	—	—	10
									20
									6
									10
									10
										16ビット転送
										SP←ソース	

LD IX, nn′ LD IX,(nn′)	DD 21 n′ n DD 2A n′ n	—	—	—	—	—	—	—	14 20	16ビット転送 IX←ソース	

ニーモニック	マシン語	フラグ変化							所要クロックサイクル	動作
		S	Z	H	P/V P	P/V V	N	C		
LD IY, nn′ LD IY,(nn′)	FD 21 n′n FD 2A n′n	—	—	—	—	—	—	—	14 20	16ビット転送 IY←ソース

LD (nn′), BC
LD (nn′), DC
LD (nn′), HL
LD (nn′), SP
LD (nn′), IX
LD (nn′), IY
	ED 43 n′n
	ED 53 n′n
	22 n′n
	ED 73 n′n
	DD 22 n′n
	FD 22 n′n
		—	—	—	—	—	—	—	20
									20
									16
									20
									20
									20
										16ビット転送（ロード）
										(nn′)←ソースL
										(nn′＋1)←ソースH

LDD	ED A8	×	×	0	—	•	0	—	16	ブロック転送 （ロード・ディクリメント） (DE)←(HL) DE←DE－1 HL←HL－1 BC←BC－1
LDDR	ED B8	×	×	0	—	0	0	—	1バイトにつき 21 最終のみ 16	ブロック転送 （ロード・ディクリメント・リピート） LDDをBC＝0までくり返す
LDI	ED A0	×	×	0	—	•	0	—	16	ブロック転送 （ロード・インクリメント） (DE)←(HL) DE←DE＋1 HL←HL＋1 BC←BC－1
LDIR	ED B0	×	×	0	—	0	0	—	1バイトにつき 21 最終のみ 16	ブロック転送 （ロード・インクリメント・リピート） LDIをBC＝0までくり返す
NEG	ED 44	•	•	•	—	•	1	•	8	ニゲイト 2の補数をとる A←0－A
NOP	00	—	—	—	—	—	—	—	4	何もしないで次へ （ノーオペレーション）

ニーモニック	マシン語	フラグ変化 S	Z	H	P/V P	V	N	C	所要クロックサイクル	動作
OR n	F6 n								7	論理和（オア）
OR A OR B OR C OR D OR E OR H OR L	B7 B0 B1 B2 B3 B4 B5	・	・	0	・	—	0	0	4	A←A∨ソース a b 答 0 0 0 0 1 1 1 0 1 1 1 1
OR （HL）	B6								7	
OR （IX+d）	DD B6 d								19	
OR （IY+d）	FD B6 d								19	
OUT (C), A OUT (C), B OUT (C), C OUT (C), D OUT (C), E OUT (C), H OUT (C), L	ED 79 ED 41 ED 49 ED 51 ED 59 ED 61 ED 69	—	—	—	—	—	—	—	12	出力 各レジスタの内容をBCレジスタの内容番地のポートへ出力〔I/0（BC）←ソース〕
OUT (n), A	D3 n	—	—	—	—	—	—	—	11	出力 I/0（An）←A
OUTD	ED AB	×	・	×	×	—	×	×	16	アウト・ディクリメント I/0(BC)←(HL) HL←HL−1 B←B−1
OTDR	ED BB	×	1	×	×	—	×	×	1バイトにつき 21 最終のみ 16	アウト・ディクリメント・リピート OUTDをB＝0までくり返す
OUTI	ED A3	×	・	×	×	—	×	×	16	アウト・インクリメント I/0(BC)←(HL) HL←HL+1 B←B−1

ニーモニック	マシン語	S	Z	H	P/V (P)	P/V (V)	N	C	所要クロックサイクル	動作
OTIR	ED B3	×	1	×	×	−	×	×	1バイトにつき 21 最終のみ 16	アウト・インクリメント・リピート OUTIをB＝0までくり返す
POP AF	F1									16ビット転送（ポップ）
POP BC	C1								10	スタックからレジスタへ転送
POP DE	D1	—	—	—	—	—	—	—		ディスティネーションL←（SP）
POP HL	E1									ディスティネーションH←
POP IX	DD E1								14	（SP＋1）
POP IY	FD E1								14	SP←SP＋2
PUSH AF	F5									16ビット転送（プッシュ）
PUSH BC	C5								11	レジスタからスタックへ転送
PUSH DE	D5	—	—	—	—	—	—	—		（SP－1）←ソースH
PUSH HL	E5									（SP－2）←ソースL
PUSH IX	DD E5								15	SP←SP－2
PUSH IY	FD E5								15	
RES 0, A	CB 87									ビットリセット
RES 0, B	CB 80									
RES 0, C	CB 81									ソースの第0ビット←0
RES 0, D	CB 82								8	
RES 0, E	CB 83	—	—	—	—	—	—	—		
RES 0, H	CB 84									
RES 0, L	CB 85									
RES 0,(HL)	CB 86								15	
RES 0,(IX+d)	DD CB d 86								23	
RES 0,(IY+d)	FD CB d 86								23	
RES 1, A	CB 8F									ビットリセット
RES 1, B	CB 88									
RES 1, C	CB 89									ソースの第1ビット←0
RES 1, D	CB 8A								8	
RES 1, E	CB 8B	—	—	—	—	—	—	—		
RES 1, H	CB 8C									
RES 1, L	CB 8D									
RES 1,(HL)	CB 8E								15	

ニーモニック	マシン語	フラグ変化 S	Z	H	P/V P	V	N	C	所要クロックサイクル	動作
RES 1,(IX+d)	DD CB d 8E								23	
RES 1,(IY+d)	FD CB d 8E								23	
RES 2, A	CB 97									ビットリセット
RES 2, B	CB 90									
RES 2, C	CB 91									ソースの第2ビット←0
RES 2, D	CB 92								8	
RES 2, E	CB 93	—	—	—	—	—	—	—		
RES 2, H	CB 94									
RES 2, L	CB 95									
RES 2,(HL)	CB 96								15	
RES 2,(IX+d)	DD CB d 96								23	
RES 2,(IY+d)	FD CB d 96								23	
RES 3, A	CB 9F									ビットリセット
RES 3, B	CB 98									
RES 3, C	CB 99									ソースの第3ビット←0
RES 3, D	CB 9A								8	
RES 3, E	CB 9B	—	—	—	—	—	—	—		
RES 3, H	CB 9C									
RES 3, L	CB 9D									
RES 3,(HL)	CB 9E								15	
RES 3,(IX+d)	DD CB d 9E								23	
RES 3,(IY+d)	FD CB d 9E								23	
RES 4, A	CB A7									ビットリセット
RES 4, B	CB A0									
RES 4, C	CB A1									ソースの第4ビット←0
RES 4, D	CB A2								8	
RES 4, E	CB A3	—	—	—	—	—	—	—		
RES 4, H	CB A4									
RES 4, L	CB A5									
RES 4,(HL)	CB A6								15	
RES 4,(IX+d)	DD CB d A6								23	
RES 4,(IY+d)	FD CB d A6								23	
RES 5, A	CB AF									ビットリセット
RES 5, B	CB A8									

ニーモニック	マシン語	S	Z	H	P/V	N	C	所要クロックサイクル	動作
RES 5, C	CB A9								ソースの第5ビット←0
RES 5, D	CB AA							8	
RES 5, E	CB AB	—	—	—	—	—	—		
RES 5, H	CB AC								
RES 5, L	CB AD								
RES 5,(HL)	CB AE							15	
RES 5,(IX+d)	DD CB d AE							23	
RES 5,(IY+d)	FD CB d AE							23	
RES 6, A	CB B7								ビットリセット
RES 6, B	CB B0								
RES 6, C	CB B1								ソースの第6ビット←0
RES 6, D	CB B2							8	
RES 6, E	CB B3	—	—	—	—	—	—		
RES 6, H	CB B4								
RES 6, L	CB B5								
RES 6,(HL)	CB B6							15	
RES 6,(IX+d)	DD CB d B6							23	
RES 6,(IY+d)	FD CB d B6							23	
RES 7, A	CB BF								ビットリセット
RES 7, B	CB B8								
RES 7, C	CB B9								ソースの第7ビット←0
RES 7, D	CB BA							8	
RES 7, E	CB BB	—	—	—	—	—	—		
RES 7, H	CB BC								
RES 7, L	CB BD								
RES 7,(HL)	CB BE							15	
RES 7,(IX+d)	DD CB d BE							23	
RES 7,(IY+d)	FD CB d BE							23	
RET	C9	—	—	—	—	—	—	10	サブルーチンからのリターン PCへスタックよりPOP
RET NZ	C0								条件付リターン
RET Z	C8							条件成立	・条件が成立すれば
RET NC	D0							11	POP PC

ニーモニック	マシン語	フラグ変化 S	Z	H	P/V P	V	N	C	所要クロックサイクル	動作
RET C	D8	—	—	—	—	—	—	—	不成立 5	・不成立なら本命令は無視する
RET PO	E0									
RET PE	E8									
RET P	F0									
RET M	F8									
RETI	ED 4D	—	—	—	—	—	—	—	14	割り込み処理からのリターン POP PC
RETN	ED 45	—	—	—	—	—	—	—	14	ノンマスカブル割り込み処理からのリターン POP PC
RLA	17	—	—	0	—	—	0	•	4	ローテート・レフト・アキュムレーター RL Aと同じ
RLCA	07	—	—	0	—	—	0	•	4	ローテート・レフト・サーキュラ・アキュムレーター RLC Aと同じ
RRA	1F	—	—	0	—	—	0	•	4	ローテート・ライト・アキュムレーター RR Aと同じ
RRCA	OF	—	—	0	—	—	0	•	4	ローテート・ライト・サーキュラ・アキュムレーター RRC Aと同じ
RL A	CB 17	•	•	0	•	—	0	•	8	ローテート・レフト Cy ← 7 0 ←
RL B	CB 10								8	
RL C	CB 11								8	
RL D	CB 12								8	
RL E	CB 13								8	
RL H	CB 14								8	
RL L	CB 15								8	
RL (HL)	CB 16								15	
RL (IX+d)	DD CB d 16								23	
RL (IY+d)	FF CB d 16								23	

ニーモニック	マシン語	S	Z	H	P/V P	P/V V	N	C	所要クロックサイクル	動作
RLC A	CB 07									ローテート・レフト・サーキュラ
RLC B	CB 00									
RLC C	CB 01									
RLC D	CB 02								8	
RLC E	CB 03	・	・	0	・	−	0	・		Cy ← 7 0
RLC H	CB 04									
RLC L	CB 05									
RLC （HL）	CB 06								15	
RLC （IX+d）	DD CB d 06								23	
RLC （IY+d）	FD CB d 06								23	
RR A	CB 1F									ローテート・ライト
RR B	CB 18									
RR C	CB 19									
RR D	CB 1A								8	
RR E	CB 1B	・	・	0	・	−	0	・		Cy → 7 0
RR H	CB 1C									
RR L	CB 1D									
RR （HL）	CB 1E								15	
RR （IX+d）	DD CB d 1E								23	
RR （IY+d）	FD CB d 1E								23	
RRC A	CB 0F									ローテート・ライト・サーキュラ
RRC B	CB 08									
RRC C	CB 09									
RRC D	CB 0A								8	
RRC E	CB 0B	・	・	0	・	−	0	・		Cy 7 0
RRC H	CB 0C									
RRC L	CB 0D									
RRC （HL）	CB 0E								15	
RRC （IX+d）	DD CB d 0E								23	
RRC （IY+d）	FD CB d 0E								23	
RLD	ED 6F	・	・	0	・	−	0	−	18	ローテート・レフト・ディジット A (HL) 7 4 3 0 7 4 3 0

ニーモニック	マシン語	S	Z	H	P/V P	P/V V	N	C	所要クロックサイクル	動作
RRD	ED 67	・	・	0	・	ー	0	ー	18	ローテート・ライト・ディジット A (HL) 7 4 3 0 7 4 3 0
RST 00H	C7	ー	ー	ー	ー	ー	ー	ー	11	リスタート 0000 H～0038 Hのいずれかの番地に対するCALL
RST 08H	CF									
RST 10H	D7									
RST 18H	DF									
RST 20H	E7									
RST 28H	EF									
RST 30H	F7									
RST 38H	FF									
SBC A, n	DE n	・	・	・	ー	・	1	・	7	8ビット引き算（キャリ付） （サブトラクト・ウィズ・キャリ） A←A－ソース－Cy
SBC A, A	9F								4	
SBC A, B	98								4	
SBC A, C	99								4	
SBC A, D	9A								4	
SBC A, E	9B								4	
SBC A, H	9C								4	
SBC A, L	9D								4	
SBC A,(HL)	9E								7	
SBC A,(IX+d)	DD 9E d								19	
SBC A,(IY+d)	FD 9E d								19	
SBC HL, BC	ED 42	・	・	×	ー	・	1	・	15	16ビット引き算（キャリ付） （サブトラクト・ウィズ・キャリ） HL←HL－ソース－Cy
SBC HL, DE	ED 52									
SBC HL, HL	ED 62									
SBC HL, SP	ED 72									
SCF	37	ー	ー	0	ー	ー	0	1	4	セット・キャリフラグ Cy←1
SET 0, A	CB C7									ビットセット
SET 0, B	CB C0									
SET 0, C	CB C1									

ニーモニック	マシン語	フラグ変化 S	Z	H	P/V P	V	N	C	所要クロックサイクル	動作
SET 0, D	CB C2								8	ソースの第0ビット←1
SET 0, E	CB C3	—	—	—	—	—	—	—		
SET 0, H	CB C4									
SET 0, L	CB C5									
SET 0, (HL)	CB C6								15	
SET 0, (IX+d)	DD CB d C6								23	
SET 0, (IY+d)	FD CB d C6								23	
SET 1, A	CB CF									ビットセット
SET 1, B	CB C8									
SET 1, C	CB C9									ソースの第1ビット←1
SET 1, D	CB CA								8	
SET 1, E	CB CB	—	—	—	—	—	—	—		
SET 1, H	CB CC									
SET 1, L	CB CD									
SET 1, (HL)	CB CE								15	
SET 1, (IX+d)	DD CB d CE								23	
SET 1, (IY+d)	FD CB d CE								23	
SET 2, A	CB D7									ビットセット
SET 2, B	CB D0									
SET 2, C	CB D1									ソースの第2ビット←1
SET 2, D	CB D2								8	
SET 2, E	CB D3	—	—	—	—	—	—	—		
SET 2, H	CB D4									
SET 2, L	CB D5									
SET 2, (HL)	CB D6								15	
SET 2, (IX+d)	DD CB d D6								23	
SET 2, (IY+d)	FD CB d D6								23	
SET 3, A	CB DF									ビットセット
SET 3, B	CB D8									
SET 3, C	CB D9									ソースの第3ビット←1
SET 3, D	CB DA								8	
SET 3, E	CB DB	—	—	—	—	—	—	—		
SET 3, H	CB DC									
SET 3, L	CB DD									
SET 3, (HL)	CB DE								15	

ニーモニック	マシン語	フラグ変化							所要クロックサイクル	動作
		S	Z	H	P/V		N	C		
					P	V				
SET 3,(IX+d)	DD CB d DE								23	
SET 3,(IY+d)	FD CB d DE								23	
SET 4, A	CB E7									ビットセット
SET 4, B	CB E0									
SET 4, C	CB E1									ソースの第4ビット← 1
SET 4, D	CB E2								8	
SET 4, E	CB E3	—	—	—	—	—	—	—		
SET 4, H	CB E4									
SET 4, L	CB E5									
SET 4,(HL)	CB E6								15	
SET 4,(IX+d)	DD CB d E6								23	
SET 4,(IY+d)	FD CB d E6								23	
SET 5, A	CB EF									ビットセット
SET 5, B	CB E8									
SET 5, C	CB E9									ソースの第5ビット← 1
SET 5, D	CB EA								8	
SET 5, E	CB EB	—	—	—	—	—	—	—		
SET 5, H	CB EC									
SET 5, L	CB ED									
SET 5,(HL)	CB EE								15	
SET 5,(IX+d)	DD CB d EE								23	
SET 5,(IY+d)	FD CB d EE								23	
SET 6, A	CB F7									ビットセット
SET 6, B	CB F0									
SET 6, C	CB F1									ソースの第6ビット← 1
SET 6, D	CB F2								8	
SET 6, E	CB F3	—	—	—	—	—	—	—		
SET 6, H	CB F4									
SET 6, L	CB F5									
SET 6,(HL)	CB F6								15	
SET 6,(IX+d)	DD CB d F6								23	
SET 6,(IY+d)	FD CB d F6								23	
SET 7, A	CB FF									ビットセット
SET 7, B	CB F8									

ニーモニック	マシン語	S	Z	H	P	V	N	C	所要クロックサイクル	動作
SET 7, C	CB F9									ソースの第7ビット← 1
SET 7, D	CB FA								8	
SET 7, E	CB FB	—	—	—	—	—	—	—		
SET 7, H	CB FC									
SET 7, L	CB FD									
SET 7,(HL)	CB FE								15	
SET 7,(IX+d)	DD CB d FE								23	
SET 7,(IY+d)	FD CB d FE								23	
SLA A	CB 27									シフト・レフト・アリスメチック
SLA B	CB 20									
SLA C	CB 21									
SLA D	CB 22								8	
SLA E	CB 23									
SLA H	CB 24	•	•	0	•	—	0	•		
SLA L	CB 25									
SLA (HL)	CB 26								15	
SLA (IX+d)	DD CB d 26								23	
SLA (IY+d)	FD CB d 26								23	
SRA A	CB 2F									シフト・ライト・アリスメチック
SRA B	CB 28									
SRA C	CB 29									
SRA D	CB 2A								8	
SRA E	CB 2B									
SRA H	CB 2C	•	•	0	•	—	0	•		
SRA L	CB 2D									
SRA (HL)	CB 2E								15	
SRA (IX+d)	DD CB d 2E								23	
SRA (IY+d)	FD CB d 2E								23	
SRL A	CB 3F									シフト・ライト・ロジカル
SRL B	CB 38									
SRL C	CB 39									
SRL D	CB 3A								8	
SRL E	CB 3B									
SRL H	CB 3C	•	•	0	•	—	0	•		
SRL L	CB 3D									

ニーモニック	マシン語	フラグ変化 S	Z	H	P/V P	V	N	C	所要クロックサイクル	動作
SRL (HL)	CB 3E								15	
SRL (IX+d)	DD CB d 3E								23	
SRL (IY+d)	FD CB d 3E								23	
SUB n	D6 n								7	8ビット引き算
SUB A	97									(サブトラクト)
SUB B	90									
SUB C	91									A←A－ソース
SUB D	92								4	
SUB E	93	・	・	・	ー	・	1	・		
SUB H	94									
SUB L	95									
SUB (HL)	96								7	
SUB (IX+d)	DD 96 d								19	
SUB (IY+d)	FD 96 d								19	
XOR n	EE n								7	排他的論理和
XOR A	AF									(エクスクルーシブ・オア)
XOR B	A8									
XOR C	A9									A←A ⊕ ソース
XOR D	AA								4	
XOR E	AB	・	・	0	・	ー	0	0		
XOR H	AC									
XOR L	AD									
XOR (HL)	AE								7	
XOR (IX+d)	DD AE d								19	
XOR (IY+d)	FD AE d								19	

a	b	答
0	0	0
0	1	1
1	0	1
1	1	0

付録2　１バイト符号付１６進数

（欄外は，16 進数
欄内は，10 進数）

上位＼下位	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
2	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
3	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
4	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
5	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
6	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
7	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
8	−128	−127	−126	−125	−124	−123	−122	−121	−120	−119	−118	−117	−116	−115	−114	−113
9	−112	−111	−110	−109	−108	−107	−106	−105	−104	−103	−102	−101	−100	−99	−98	−97
A	−96	−95	−94	−93	−92	−91	−90	−89	−88	−87	−86	−85	−84	−83	−82	−81
B	−80	−79	−78	−77	−76	−75	−74	−73	−72	−71	−70	−69	−68	−67	−66	−65
C	−64	−63	−62	−61	−60	−59	−58	−57	−56	−55	−54	−53	−52	−51	−50	−49
D	−48	−47	−46	−45	−44	−43	−42	−41	−40	−39	−38	−37	−36	−35	−34	−33
E	−32	−31	−30	−29	−28	−27	−26	−25	−24	−23	−22	−21	−20	−19	−18	−17
F	−16	−15	−14	−13	−12	−11	−10	−9	−8	−7	−6	−5	−4	−3	−2	−1

付録3　チェック・サム・プログラム

```
10 '████████████████████████
20 '█                      █
30 '█  ﾁｪｯｸ･ｻﾑ ﾌﾟﾛｸﾞﾗﾑ      █
40 '█                      █
50 '█     by Yasuhiro Shimizu █
60 '█                      █
70 '█     1983.10.15       █
80 '█                      █
90 '████████████████████████
100 '
110 '
120 ' ████    ｼｮｷｶ    ████
130 '
140 CLEAR &HCFF0 :OPTION BASE 0 :DEFINT M,C,D :DIM M(15,7),C(15),D(7)
150 INIT :WIDTH 40 :CLS 4 :COLOR 4
160 '
170 ' ████    ｾﾂﾒｲ    ████
180 '
190 PRINT "*** ﾁｪｯｸ･ｻﾑ ﾌﾟﾛｸﾞﾗﾑ ***" :PRINT :COLOR 3
200 PRINT "ｺﾉ ﾌﾟﾛｸﾞﾗﾑ ﾊ､ ｼﾃｲ ｻﾚﾀ ﾊﾝｲ ﾉ"
210 PRINT "ﾒﾓﾘｰ ﾉ ﾅｲﾖｳ ｦ ﾁｪｯｸ･ｻﾑ ﾋｮｳｼﾞ"
220 PRINT "ｼﾏｽ｡" :PRINT :PRINT :COLOR 6
230 PRINT "[ﾁｭｳｲ] ｾｲｶｸ ｦ ｷｽﾀﾒﾆ､ ｼﾃｲﾊﾝｲ ｲｺｳ ﾉ"
240 PRINT "         ﾒﾓﾘｰ ﾅｲﾖｳ ｦ ｽﾍﾞﾃ 0 ﾆ ｼﾏｽﾉﾃﾞ"
250 PRINT "         ﾋﾂﾖｳ ﾅﾗ､ SAVE ｼﾃｵｲﾃ ｸﾀﾞｻｲ｡" :PRINT :COLOR 2
260 PRINT "ﾖｳｲ ﾊ ｲｲﾃﾞｽｶ ? " :CFLASH 1 :COLOR 7
270 PRINT "YES = ﾘﾀｰﾝ ｷｰ, NO = ｿﾉﾀ ﾉ ｷｰ "  :CFLASH 0
280 REPEAT :I$=INKEY$ :UNTIL I$<>""
290 IF I$<>CHR$(13) THEN CLS :END
300 '
310 ' ████    ﾏｼﾝｺﾞ ｶｷｺﾐ    ████
320 '
330 RESTORE 3000
340 AD=&HCFF0      :' ﾏｼﾝｺﾞ･ｴﾘｱ = &HCFF0 ｶﾗ &HCFFD ﾏﾃﾞ
350 FOR I=0 TO 13
360   READ MC$ :POKE AD+I,VAL("&H"+MC$)
370 NEXT
380 DEF USR=&HCFF0
390 '
400 ' ████    ﾆｭｳﾘｮｸ    ████
410 '
420 CLS :COLOR 7 :CFLASH 1
430 PRINT "ADDRESS ﾊ D000 ｲｺｳ ﾆ ｼﾃｲｼﾃ ｸﾀﾞｻｲ !" :CFLASH 0 :COLOR 4 :PRINT :PRINT
440 BEEP :INPUT "START ADDRESS (16ｼﾝ 4ｹﾀ) = ";ST$ :ST=VAL("&H"+ST$) :IF ST<0 THEN ST=ST+65536!
450 IF ST<53248! THEN PRINT CHR$(7,30,5); :GOTO 440
460 COLOR 6
470 BEEP :INPUT "  END ADDRESS (16ｼﾝ 4ｹﾀ) = ";ED$ :ED=VAL("&H"+ED$) :IF ED<0 THEN ED=ED+65536!
480 IF ED<ST THEN PRINT CHR$(7,30,5); :GOTO 470
490 PRINT :PRINT :COLOR 3 :BEEP
500 PRINT "ﾃｲｾｲ ｼﾏｽｶ ? [ NO=ﾘﾀｰﾝ･ｷｰ,YES=Y ] ";
510 REPEAT :I$=INKEY$ :UNTIL I$<>""
520 IF INSTR(1,"Yyﾝ",I$)>0 THEN 420
530 LOCATE 0,20 :COLOR 5 :CFLASH 1
540 PRINT "ﾁｪｯｸ･ｻﾑ ｹｲｻﾝ ﾁｭｳ" :CFLASH 0
550 '
560 ' ████    ﾒﾓﾘｰ･ｸﾘｱ    ████
570 '
580 REM ──> ｸﾘｱ ﾊ ED+1 ｶﾗ &HFEFF ﾏﾃﾞ (&HFF00 ｶﾗ ﾊ ﾓﾆﾀｰ ﾜｰｸ･ｴﾘｱ)
590 '
600 A%=USR(CINT(ED+1))
610 '
620 ' ████    ﾒｲﾝ ﾙｰﾁﾝ    ████
630 '
640 AD=ST
650 '
660 ' ── 128 ﾊﾞｲﾄ ﾖﾐﾄﾘ ﾄ ﾖｺ ﾉ SUM ──
670 '
680 FOR I=0 TO 15
690   C(I)=0
700   FOR J=0 TO 7
710     M(I,J)=PEEK(AD+I*8+J) :C(I)=C(I)+M(I,J)
720   NEXT
730 NEXT
740 '
750 ' ── ﾀﾃ ﾉ SUM ﾄ ﾄｰﾀﾙ ──
760 '
770 S=0
780 FOR J=0 TO 7
790   D(J)=0
800   FOR I=0 TO 15
810     D(J)=D(J)+M(I,J)
```

```
820   NEXT
830   S=S+D(J)
840 NEXT
850 '
860 ' ── ﾁｪｯｸ･ｻﾑ ﾋｮｳｼﾞ ──
870 '
880 INIT :CLS :COLOR 4
890 PRINT "***  Check Sum  ***" :PRINT :COLOR 7
900 FOR I=0 TO 15
910   PRINT ":"+RIGHT$("000"+HEX$(AD+I*8),4)+" = ";
920   FOR J=0 TO 7
930     PRINT  RIGHT$("0"+HEX$(M(I,J)),2)+" ";
940   NEXT
950   PRINT " : ";RIGHT$("0"+HEX$(C(I)),2)
960 NEXT
970 PRINT
980 FOR I=1 TO 37 :PRINT "-"; :NEXT :PRINT
990 PRINT "           ";
1000 FOR J=0 TO 7
1010   PRINT RIGHT$("0"+HEX$(D(J)),2)+" ";
1020 NEXT
1030 PRINT " : ";RIGHT$("0"+HEX$(S),2)
1040 '
1050 ' ── ﾃｲｾｲ ──
1060 '
1070 CONSOLE 23,2 :CLS :COLOR 6
1080 BEEP :PRINT "ﾃｲｾｲ ｶｼｮ ﾊ ｱﾘﾏｽｶ ? [ NO=ﾘﾀｰﾝ･ｷｰ YES=Y ] ";
1090 REPEAT :I$=INKEY$ :UNTIL I$<>""
1100 IF INSTR(1,"Yyﾝ",I$)=0 THEN 1230
1110 ADR$="" :BEEP :COLOR 3 :PRINT :INPUT "ﾃｲｾｲ･ｱﾄﾞﾚｽ  ﾊ [ 16ｼﾝ 4ｹﾀ ] ";ADR$
1120 ADR=VAL("&H"+ADR$) :IF ADR<0 THEN ADR=ADR+65536!
1130 IF ADR<AD OR ADR>(AD+127) THEN CLS :BEEP :GOTO 1110
1140 MC$=RIGHT$("0"+HEX$(PEEK(ADR)),2)
1150 MC1$="" :BEEP :COLOR 2 :PRINT ADR$+" = "+MC$+" -> "; :INPUT MC1$
1160 IF MC1$="" THEN 1070
1170 POKE ADR,VAL("&H"+MC1$)
1180 COLOR 6 :CFLASH 1 :PRINT "ﾁｪｯｸ･ｻﾑ ｹｲｻﾝ ﾁｭｳ"; :CFLASH 0
1190 GOTO 680  :'──> ﾁｪｯｸ･ｻﾑ ｹｲｻﾝ
1200 '
1210 ' ── ﾂｷﾞ ﾉ ｼﾞｭﾝﾋﾞ ﾄ ｵﾜﾘ ﾉ ﾊﾝﾃｲ ──
1220 '
1230 BEEP :COLOR 4 :PRINT :PR=0
1240 PRINT "ﾌﾟﾘﾝﾄ ｼﾏｽｶ [ NO=ﾘﾀｰﾝ･ｷｰ,YES=1 ] "; :I$=INKEY$(1) :PR=VAL(I$)
1250 IF PR=1 THEN GOSUB 2000
1260 '
1270 AD=AD+128
1280 IF AD<=ED THEN MUSIC "O5G3+C" :COLOR 7 :CLS :CFLASH 1 :PRINT "ﾂｷﾞ ﾆ ｽｽﾐﾏｽ!  ｹｲｻﾝ ﾁｭｳ｡"; :CF
LASH 0 :GOTO 680
1290 '
1300 CONSOLE 22,3 :CLS :INIT :COLOR 3
1310 PRINT "ｵﾜﾘﾏｼﾀ｡ "
1320 MUSIC "O4+C3BAGFEDC" :COLOR 7
1330 END
1340 '
1350 '
2000 ' ■■■■    ﾌﾟﾘﾝﾄ ｱｳﾄ ｻﾌﾞﾙｰﾁﾝ    ■■■■
2010 '
2020 FOR I=0 TO 15
2030   LPRINT ":"+RIGHT$("000"+HEX$(AD+I*8),4)+" = ";
2040   FOR J=0 TO 7
2050     LPRINT  RIGHT$("0"+HEX$(M(I,J)),2)+" ";
2060   NEXT
2070   LPRINT " : ";RIGHT$("0"+HEX$(C(I)),2)
2080 NEXT
2090 LPRINT
2100 FOR I=1 TO 37 :LPRINT "-"; :NEXT :LPRINT
2110 LPRINT "           ";
2120 FOR J=0 TO 7
2130   LPRINT RIGHT$("0"+HEX$(D(J)),2)+" ";
2140 NEXT
2150 LPRINT " : ";RIGHT$("0"+HEX$(S),2) :LPRINT :LPRINT
2160 RETURN
2170 '
2180 '
2980 ' ■■■■    ﾏｼﾝｺﾞ ﾃﾞｰﾀ    ■■■■
2990 '
3000 DATA 5E,23,56,D5,E1,36,00,23
3010 DATA 7C,FE,FF,38,F8,C9
3020 '
3030 ' ■■■■■■■■■■■■■■■■
```

付録4　マシン語　DATA　ジェネレータ

```
10 '■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
20 '■                            ■
30 '■  ﾏｼﾝｺﾞ DATA ｼﾞｪﾈﾚｰﾀｰ        ■
40 '■                            ■
50 '■       by Yasuhiro Shimizu  ■
60 '■                            ■
70 '■       1983.10.15           ■
80 '■                            ■
90 '■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
100 '
110 '
120 ' ■■■■■   ｼｮｷｶ   ■■■■■
130 '
140 CLEAR &HD000
150 KBUF ON
160 INIT :CLS 4 :WIDTH 40 :COLOR 4
170 '
180 ' ■■■■■   ｾﾂﾒｲ   ■■■■■
190 '
200 PRINT "ｺﾉ ﾌﾟﾛｸﾞﾗﾑ ﾊ､ ﾒﾓﾘｰ D000H ﾊﾞﾝﾁ ｲｺｳ ﾆ"
210 PRINT "ｶｶﾚﾀ､ ﾏｼﾝｺﾞﾌﾟﾛｸﾞﾗﾑ ｦ BASIC ﾉ DATA ﾌﾞﾝ"
220 PRINT "ﾄｼﾃ､ ｷﾞｮｳﾊﾞﾝｺﾞｳ 30000 ｲｺｳ ﾆ ﾂｸﾘﾏｽ｡"
230 LOCATE 0,5 :COLOR 6
240 PRINT "Address ｦ ﾆｭｳﾘｮｸ ｽﾙﾄ､ DATA ﾌﾞﾝ ｦ ｼﾞﾄﾞｳ"
250 PRINT "ｾｲｾｲ ｼﾏｽ｡ "
260 '
270 ' ■■■■■   ﾆｭｳﾘｮｸ   ■■■■■
280 '
290 LOCATE 0,10 :COLOR 3
300 INPUT "ﾏｼﾝｺﾞ  START ADDRESS (16ｼﾝ 4ｹﾀ) = ";ST$ :ST=VAL("&H"+ST$) :IF ST<0 THEN ST=ST+65536!
310 IF ST<53248! THEN PRINT CHR$(7,&H1E,5); :GOTO 300
320 COLOR 2
330 INPUT "ﾏｼﾝｺﾞ  END  ADDRESS (16ｼﾝ 4ｹﾀ) = ";ED$ :ED=VAL("&H"+ED$) :IF ED<0 THEN ED=ED+65536!
340 IF ED<ST THEN PRINT CHR$(7,&H1E,5); :GOTO 330
350 LOCATE 0,15 :COLOR 7
360 PRINT "ﾃｲｾｲ ｼﾏｽｶ ? [ Y or N ] ";
370 I$=INKEY$(1)
380 IF INSTR(1,"Yyﾝ",I$)>0 THEN 140 ELSE IF INSTR(1,"Nnﾐ",I$)>0 THEN 400 ELSE 370
390 '
400 ' ■■■■■   ﾒｲﾝ ﾙｰﾁﾝ   ■■■■■
410 '
420 AD=ST :GYO=30000
430 '
440 IF (AD+7)>ED THEN 590
450   MC$=""
460   FOR I=0 TO 7
470     IF I<7 THEN 480 ELSE 490
480     MC$=MC$+RIGHT$("0"+HEX$(PEEK(AD)),2)+"," :GOTO 500
490     MC$=MC$+RIGHT$("0"+HEX$(PEEK(AD)),2)
500     AD=AD+1
510   NEXT
520   D$="DATA "+MC$
530   FLAG=0 :GOTO "ｼﾞﾄﾞｳｾｲｾｲ"
540   GYO=GYO+10
550 GOTO 440
560 '
570 ' ―――――   ｵﾜﾘ ｼｮﾘ   ―――――
580 '
590 MC$=""
600 L=ED-AD :IF L<0 THEN 700
610 FOR I=0 TO L
620   IF I=L THEN 640
630   MC$=MC$+RIGHT$("0"+HEX$(PEEK(AD)),2)+"," :GOTO 650
640   MC$=MC$+RIGHT$("0"+HEX$(PEEK(AD)),2)
650   AD=AD+1
660 NEXT
670 D$="DATA "+MC$
680 FLAG=1 :GOTO "ｼﾞﾄﾞｳｾｲｾｲ"
690 '
700 CONSOLE 0,20 :CLS :CONSOLE :COLOR 7 :MUSIC "O5G2+C"
710 PRINT "DATA ﾌﾞﾝ ｶﾞ ｶﾝｾｲ ｼﾏｼﾀ｡"
720 PRINT "DELETE -900 ｦ ｼﾞｯｺｳｼﾃ"
730 PRINT "ﾎﾝﾀｲ･ﾌﾟﾛｸﾞﾗﾑ ｦ ｹｼﾃｸﾀﾞｻｲ !"
740 PRINT
750 END
760 '
770 '     ■■■■■■■■■■
780 '
790 LABEL "ｼﾞﾄﾞｳｾｲｾｲ"
800 '
810 '     ■■■■■■■■■■
```

```
820
830 CONSOLE 22,3
840 LOCATE 0,24 :COLOR 6
850 PRINT CHR$(5);
860 PRINT GYO;D$:PRINT
870 LOCATE 0,23 :COLOR 0
880 KEY0,CHR$(30,30,13)+"GOTO890"+CHR$(13) :END
890 IF FLAG=0 THEN 540
900 GOTO 690
```

付録5 TRON GAME マシン語ソースリスト（「ミニアセンブラ」使用）

```
Page 00001

00010 0000                               #3
00021 0000                                                        ;
00022 0000                                                        ; ■■■■■■■■■■■■■■■■■■■■
00023 0000                                                        ;
00024 0000                                                        ; M.L. Subs of TRON GAME
00025 0000                                                        ;
00026 0000                                                        ;  by Y. Shimizu  1983.10.15
00027 0000                                                        ;
00028 0000                                                        ; ■■■■■■■■■■■■■■■■■■■■
00029 0000                                                        ;
00030 0000                               ORG    $D000
00040 D000                                                        ;
00050 D000                                                        ; ■ working area ■
00060 D000                                                        ;
00070 D000               TRX:                   EQU $E000         ; X coord of TRON
00080 D000               TRY:                   EQU $E001         ; Y coord of TRON
00090 D000               SKX:                   EQU $E002         ; X coord of SARK
00100 D000               SKY:                   EQU $E003         ; Y coord of SARK
00110 D000               TRMV:                  EQU $E004         ; move code of TRON
00120 D000               SKMV:                  EQU $E005         ; move code of SARK
00130 D000               RND:                   EQU $E006         ; random
00140 D000               FLAG:                  EQU $E007         ; cont flag (0=cont,1=end)
00150 D000               ADR:                   EQU $E008         ; address work (2 bytes)
00160 D000               NADR:                  EQU $E00A         ; new address work (2 bytes)
00170 D000               MOVE:                  EQU $E00C         ; move code work
00180 D000               TS:                    EQU $E00D         ; TRON SARK flag (0=TRON)
00190 D000               ORBIT:                 EQU $E00E         ; character of orbit
00200 D000               BIKE:                  EQU $E00F         ; character of bike
00210 D000                                                        ;
00220 D000                                                        ; ■ Sub of SARK move ■
00230 D000                                                        ;
00240 D000 3E01          SARKMV:                LD A,1            ; SARK flag
00250 D002 320DE0                        LD     (TS),A
00260 D005 CDECD0                        CALL   ADCAL
00270 D008 3A06E0                        LD     A,(RND)
00280 D00B FE04                          CP     4
00290 D00D DA44D0                        JP     C,SELF
00300 D010                                                        ;
00310 D010 3A01E0        CPTRON:                LD A,(TRY)
00320 D013 2103E0                        LD     HL,SKY
00330 D016 BE                            CP     (HL)              ; (TRY) < (SKY) ?
00340 D017 DC1BD1                        CALL   C,GOUP
00350 D01A DAF7D1                        JP     C,CHRPR
00360 D01D                                                        ;
00370 D01D 3A01E0                        LD     A,(TRY)
00380 D020 2103E0                        LD     HL,SKY
00390 D023 BE                            CP     (HL)              ; (TRY) > (SKY) ?
00400 D024 C458D1                        CALL   NZ,GODN
00410 D027 DAF7D1                        JP     C,CHRPR
00420 D02A                                                        ;
```

```
00430 D02A 3A00E0                    LD    A,(TRX)
00440 D02D 2102E0                    LD    HL,SKX
00450 D030 BE                        CP    (HL)         ; (TRX) < (SKX) ?
00460 D031 DC91D1                    CALL  C,GOLT
00470 D034 DAF7D1                    JP    C,CHRPR
00480 D037                                              ;
00490 D037 3A00E0                    LD    A,(TRX)
00500 D03A 2102E0                    LD    HL,SKX
00510 D03D BE                        CP    (HL)         ; (TRX) > (SKX) ?
00520 D03E C4C4D1                    CALL  NZ,GORT
00530 D041 DAF7D1                    JP    C,CHRPR
00540 D044                                              ;
00550 D044 CD1BD1          SELF:           CALL GOUP
00560 D047 DAF7D1                    JP    C,CHRPR
00570 D04A CD58D1                    CALL  GODN
00580 D04D DAF7D1                    JP    C,CHRPR
00590 D050 CD91D1                    CALL  GOLT
00600 D053 DAF7D1                    JP    C,CHRPR
00610 D056 CDC4D1                    CALL  GORT
00620 D059 DAF7D1                    JP    C,CHRPR
00630 D05C                                              ;
00640 D05C                                              ; Crash of SARK
00650 D05C                                              ;
00660 D05C 3A0CE0                    LD    A,(MOVE)
00670 D05F FE08                      CP    8            ;
00680 D061 2005                      JR    NZ,NEXT2
00690 D063 CD2ED1                    CALL  GOUP1        ;
00700 D066 1815                      JR    CRHSK
00710 D068 FE02            NEXT2:          CP 2         ;
00720 D06A 2005                      JR    NZ,NEXT4
00730 D06C CD69D1                    CALL  GODN1        ;
00740 D06F 180C                      JR    CRHSK
00750 D071 FE04            NEXT4:          CP 4         ;
00760 D073 2005                      JR    NZ,NEXT6
00770 D075 CD9FD1                    CALL  GOLT1        ;
00780 D078 1803                      JR    CRHSK
00790 D07A CDD2D1          NEXT6:          CALL GORT1   ;
00800 D07D 3E01            CRHSK:          LD A,1
00810 D07F 3207E0                    LD    (FLAG),A     ; crash flag set for SARK
00820 D082 C3F7D1                    JP    CHRPR
00830 D085                                              ;
00840 D085                                              ; ■■ Sub of TRON move ■■
00850 D085                                              ;
00860 D085 3E00            TRONMV:         LD A,0       ; TRON flag
00870 D087 320DE0                    LD    (TS),A
00880 D08A CDECD0                    CALL  ADCAL
00890 D08D                                              ;
00900 D08D 3A04E0                    LD    A,(TRMV)
00910 D090 210CE0                    LD    HL,MOVE
00920 D093 86                        ADD   A,(HL)
```

```
00930 D094 FE0A                CP     10
00940 D096 2003                JR     NZ,KYSCAN
00950 D098                                              ;
00960 D098 2104E0    GODR:           LD HL,TRMV         ; Go direct
00970 D09B                                              ;
00980 D09B 7E        KYSCAN:         LD A,(HL)          ; Keyscan
00990 D09C FE08      KEY8:           CP 8
01000 D09E 2010                JR     NZ,KEY2
01010 D0A0 CD1BD1              CALL   GOUP
01020 D0A3 3808                JR     C,EXIT8
01030 D0A5 CD2ED1              CALL   GOUP1
01040 D0A8 3E01                LD     A,1
01050 D0AA 3207E0              LD     (FLAG),A
01060 D0AD C3F7D1    EXIT8:          JP CHRPR
01070 D0B0                                              ;
01080 D0B0 FE02      KEY2:           CP 2
01090 D0B2 2010                JR     NZ,KEY4
01100 D0B4 CD58D1              CALL   GODN
01110 D0B7 3808                JR     C,EXIT2
01120 D0B9 CD69D1              CALL   GODN1
01130 D0BC 3E01                LD     A,1
01140 D0BE 3207E0              LD     (FLAG),A
01150 D0C1 C3F7D1    EXIT2:          JP CHRPR
01160 D0C4                                              ;
01170 D0C4 FE04      KEY4:           CP 4
01180 D0C6 2010                JR     NZ,KEY6
01190 D0C8 CD91D1              CALL   GOLT
01200 D0CB 3808                JR     C,EXIT4
01210 D0CD CD9FD1              CALL   GOLT1
01220 D0D0 3E01                LD     A,1
01230 D0D2 3207E0              LD     (FLAG),A
01240 D0D5 C3F7D1    EXIT4:          JP CHRPR
01250 D0D8                                              ;
01260 D0D8 FE06      KEY6:           CP 6
01270 D0DA 20BC                JR     NZ,GODR
01280 D0DC CDC4D1              CALL   GORT
01290 D0DF 3808                JR     C,EXIT6
01300 D0E1 CDD2D1              CALL   GORT1
01310 D0E4 3E01                LD     A,1
01320 D0E6 3207E0              LD     (FLAG),A
01330 D0E9 C3F7D1    EXIT6:          JP CHRPR
01340 D0EC                                              ;
01350 D0EC                                              ; ■ Subs of movement ■
01360 D0EC                                              ;
01370 D0EC 3A0DE0    ADCAL:          LD A,(TS)          ; Address calculation ▬
01380 D0EF FE00                CP     0
01390 D0F1 200C                JR     NZ,SKPART
01400 D0F3                                              ;
01410 D0F3 7E        TRPART:         LD A,(HL)          ; (HL) = Ten Key
01420 D0F4 320CE0              LD     (MOVE),A
```

```
01430 D0F7 2100E0                    LD   HL,TRX
01440 D0FA 56                        LD   D,(HL)          ; D <- (TRX)
01450 D0FB 23                        INC  HL
01460 D0FC 5E                        LD   E,(HL)          ; E <- (TRY)
01470 D0FD 180C                      JR   XFER
01480 D0FF                                                ;
01490 D0FF 3A05E0       SKPART:           LD A,(SKMV)
01500 D102 320CE0                    LD   (MOVE),A
01510 D105 2102E0                    LD   HL,SKX
01520 D108 56                        LD   D,(HL)          ; D <- (SKX)
01530 D109 23                        INC  HL
01540 D10A 5E                        LD   E,(HL)          ; E <- (SKY)
01550 D10B                                                ;
01560 D10B 210030       XFER:             LD HL,$3000     ; HL <- VRAM Top address
01570 D10E 4A                        LD   C,D
01590 D10F 1600                      LD   D,0
01600 D111 0628                      LD   B,40
01610 D113 19           LOOP1:            ADD HL,DE
01620 D114 10FD                      DJNZ LOOP1
01630 D116 09                        ADD  HL,BC           ; HL <- HL+E*40+D
01640 D117 2208E0                    LD   (ADR),HL
01650 D11A C9                        RET
01660 D11B                                                ;
01670 D11B 2A08E0       GOUP:             LD HL,(ADR)     ; Go up ?
01680 D11E 112800                    LD   DE,40
01690 D121 B7                        OR   A
01700 D122 ED52                      SBC  HL,DE           ; HL <- HL-40
01710 D124 44                        LD   B,H
01720 D125 4D                        LD   C,L
01730 D126 ED78                      IN   A,(C)
01740 D128 FE20                      CP   $20             ; up ?
01750 D12A 280B                      JR   Z,GOUP2
01760 D12C B7                        OR   A               ; Cy <- 0
01770 D12D C9                        RET
01780 D12E                                                ;
01790 D12E 2A08E0       GOUP1:            LD HL,(ADR)     ; Go up !
01800 D131 112800                    LD   DE,40
01810 D134 B7                        OR   A
01820 D135 ED52                      SBC  HL,DE
01830 D137 220AE0       GOUP2:            LD (NADR),HL
01840 D13A 3E08                      LD   A,8             ; up code = 8
01850 D13C 320CE0                    LD   (MOVE),A
01860 D13F 3A0DE0                    LD   A,(TS)
01870 D142 FE00                      CP   0
01880 D144 2009                      JR   NZ,SKUP
01890 D146                                                ;
01900 D146 3A01E0       TRUP:             LD A,(TRY)      ; Go up for TRON
01910 D149 3D                        DEC  A
01920 D14A 3201E0                    LD   (TRY),A         ; (TRY) <- (TRY)-1
01930 D14D 1807                      JR   UPFLG
```

```
01940 D14F
01950 D14F 3A03E0       SKUP:        LD A,(SKY)        ; Go up for SARK
01960 D152 3D                   DEC    A
01970 D153 3203E0               LD     (SKY),A         ; (SKY) <- (SKY)-1
01980 D156 37           UPFLG:  SCF
01990 D157 C9                   RET
02000 D158                                             ;
02010 D158 2A08E0       GODN:        LD HL,(ADR)       ; Go down ?
02020 D15B 112800               LD     DE,40
02030 D15E 19                   ADD    HL,DE           ; HL <- HL+40
02040 D15F 44                   LD     B,H
02050 D160 4D                   LD     C,L
02060 D161 ED78                 IN     A,(C)
02070 D163 FE20                 CP     $20             ; down ?
02080 D165 2809                 JR     Z,GODN2
02090 D167 B7                   OR     A               ; Cy <- 0
02100 D168 C9                   RET
02110 D169                                             ;
02120 D169 2A08E0       GODN1:       LD HL,(ADR)       ; Go down !
02130 D16C 112800               LD     DE,40
02140 D16F 19                   ADD    HL,DE
02150 D170 220AE0       GODN2:       LD (NADR),HL
02160 D173 3E02                 LD     A,2             ; down code = 2
02170 D175 320CE0               LD     (MOVE),A
02180 D178 3A0DE0               LD     A,(TS)
02190 D17B FE00                 CP     0
02200 D17D 2009                 JR     NZ,SKDN
02210 D17F                                             ;
02220 D17F 3A01E0       TRDN:        LD A,(TRY)
02230 D182 3C                   INC    A
02240 D183 3201E0               LD     (TRY),A         ; (TRY) <- (TRY)+1
02250 D186 1807                 JR     DNFLG
02260 D188                                             ;
02270 D188 3A03E0       SKDN:        LD A,(SKY)
02280 D18B 3C                   INC    A
02290 D18C 3203E0               LD     (SKY),A         ; (SKY) <- (SKY)+1
02300 D18F 37           DNFLG:  SCF
02310 D190 C9                   RET
02320 D191                                             ;
02330 D191 2A08E0       GOLT:        LD HL,(ADR)       ; Go left ?
02340 D194 2B                   DEC    HL              ; HL <- HL-1
02350 D195 44                   LD     B,H
02360 D196 4D                   LD     C,L
02370 D197 ED78                 IN     A,(C)
02380 D199 FE20                 CP     $20             ; left ?
02390 D19B 2806                 JR     Z,GOLT2
02400 D19D B7                   OR     A               ; Cy <- 0
02410 D19E C9                   RET
02420 D19F                                             ;
02430 D19F 2A08E0       GOLT1:       LD HL,(ADR)       ; Go left !
```

```
02440 D1A2 2B                DEC    HL
02450 D1A3 220AE0   GOLT2:          LD (NADR),HL
02460 D1A6 3E04              LD     A,4                  ; left code = 4
02470 D1A8 320CE0            LD     (MOVE),A
02480 D1AB 3A0DE0            LD     A,(TS)
02490 D1AE FE00              CP     0
02500 D1B0 2009              JR     NZ,SKLT
02510 D1B2                                               ;
02520 D1B2 3A00E0   TRLT:           LD A,(TRX)
02530 D1B5 3D                DEC    A
02540 D1B6 3200E0            LD     (TRX),A              ; (TRX) <- (TRX)-1
02550 D1B9 1807              JR     LTFLG
02560 D1BB                                               ;
02570 D1BB 3A02E0   SKLT:           LD A,(SKX)
02580 D1BE 3D                DEC    A
02590 D1BF 3202E0            LD     (SKX),A              ; (SKX) <- (SKX)-1
02600 D1C2 37       LTFLG:          SCF
02610 D1C3 C9                RET
02620 D1C4                                               ;
02630 D1C4 2A08E0   GORT:           LD HL,(ADR)          ; Go right ?
02640 D1C7 23                INC    HL                   ; HL <- HL+1
02650 D1C8 44                LD     B,H
02660 D1C9 4D                LD     C,L
02670 D1CA ED78              IN     A,(C)
02680 D1CC FE20              CP     $20                  ; right ?
02690 D1CE 2806              JR     Z,GORT2
02700 D1D0 B7                OR     A                    ; Cy <- 0
02710 D1D1 C9                RET
02720 D1D2                                               ;
02730 D1D2 2A08E0   GORT1:          LD HL,(ADR)          ; Go right !
02740 D1D5 23                INC    HL
02750 D1D6 220AE0   GORT2:          LD (NADR),HL
02760 D1D9 3E06              LD     A,6                  ; right code = 6
02770 D1DB 320CE0            LD     (MOVE),A
02780 D1DE 3A0DE0            LD     A,(TS)
02790 D1E1 FE00              CP     0
02800 D1E3 2009              JR     NZ,SKRT
02810 D1E5                                               ;
02820 D1E5 3A00E0   TRRT:           LD A,(TRX)
02830 D1E8 3C                INC    A
02840 D1E9 3200E0            LD     (TRX),A              ; (TRX) <- (TRX)+1
02850 D1EC 1807              JR     RTFLG
02860 D1EE                                               ;
02870 D1EE 3A02E0   SKRT:           LD A,(SKX)
02880 D1F1 3C                INC    A
02890 D1F2 3202E0            LD     (SKX),A              ; (SKX) <- (SKX)+1
02900 D1F5 37       RTFLG:          SCF
02910 D1F6 C9                RET
02920 D1F7                                               ;
02930 D1F7                                               ;     Character print
```

```
02940 D1F7                                                   ;
02950 D1F7 CD26D2       CHRPR:           CALL CHRDET
02960 D1FA                                                   ;
02970 D1FA ED4B08E0              LD      BC,(ADR)
02980 D1FE 1600                  LD      D,0                 ; orbit character select
02990 D200 3A0EE0                LD      A,(ORBIT)
03000 D203 5F                    LD      E,A
03010 D204 CD3CD3                CALL    PRINT
03020 D207                                                   ;
03030 D207 3A07E0                LD      A,(FLAG)
03040 D20A FE00                  CP      0
03050 D20C 200D                  JR      NZ,CRHCHR
03060 D20E                                                   ;
03070 D20E ED4B0AE0     BIKCHR:          LD BC,(NADR)
03080 D212 1601                  LD      D,1                 ; bike character select
03090 D214 3A0FE0                LD      A,(BIKE)
03100 D217 5F                    LD      E,A
03110 D218 C33CD3                JP      PRINT
03120 D21B                                                   ;
03130 D21B ED4B0AE0     CRHCHR:          LD BC,(NADR)
03140 D21F 1602                  LD      D,2                 ; crash character select
03150 D221 1EE8                  LD      E,$E8               ; E <- "X"
03160 D223 C33CD3                JP      PRINT
03170 D226                                                   ;
03180 D226                                                   ; ■■ Subs of character ■■
03190 D226                                                   ;
03200 D226 3A0CE0       CHRDET:          LD A,(MOVE)         ; determine char ▬▬
03210 D229                                                   ;
03220 D229 FE02         MVSCAN:          CP 2                ; move scan ▬▬▬
03221 D22B CA79D2                JP      Z,CHRDN
03222 D22E FE04                  CP      4
03223 D230 CABAD2                JP      Z,CHRLT
03224 D233 FE06                  CP      6
03225 D235 CAFBD2                JP      Z,CHRRT
03230 D238                                                   ;
03240 D238 3A0DE0       CHRUP:           LD A,(TS)           ; character up ▬▬
03250 D23B FE00                  CP      0
03260 D23D 2005                  JR      NZ,SKUPOB
03270 D23F 3A04E0       TRUPOB:          LD A,(TRMV)
03280 D242 1803                  JR      LTUP
03290 D244 3A05E0       SKUPOB:          LD A,(SKMV)
03300 D247 FE04         LTUP:            CP 4
03310 D249 2004                  JR      NZ,RTUP
03320 D24B 3E99                  LD      A,$99               ; A <- "└"
03330 D24D 180A                  JR      UPOB
03340 D24F FE06         RTUP:            CP 6
03350 D251 2004                  JR      NZ,UPUP
03360 D253 3E98                  LD      A,$98               ; A <- "┘"
03370 D255 1802                  JR      UPOB
03380 D257 3E91         UPUP:            LD A,$91            ; A <- "|"
```

```
03390 D259 320EE0          UPOB:         LD  (ORBIT),A
03400 D25C                                                    ;
03410 D25C 3A0DE0                   LD   A, (TS)
03420 D25F FE00                     CP   0
03430 D261 200B                     JR   NZ,SKUPBK
03440 D263 3E64            TRUPBK:       LD A,100             ; TRON up bike
03450 D265 320FE0                   LD   (BIKE),A
03460 D268 3E08                     LD   A,8
03470 D26A 3204E0                   LD   (TRMV),A             ; (TRMV) <- 8
03480 D26D C9                       RET
03490 D26E 3EC8            SKUPBK:       LD A,200             ; SARK up bike
03500 D270 320FE0                   LD   (BIKE),A
03510 D273 3E08                     LD   A,8
03520 D275 3205E0                   LD   (SKMV),A             ; (SKMV) <- 8
03530 D278 C9                       RET
03540 D279                                                    ;
03570 D279 3A0DE0          CHRDN:        LD A, (TS)           ; character down
03580 D27C FE00                     CP   0
03590 D27E 2005                     JR   NZ,SKDNOB
03600 D280 3A04E0          TRDNOB:       LD A, (TRMV)
03610 D283 1803                     JR   LTDN
03620 D285 3A05E0          SKDNOB:       LD A, (SKMV)
03630 D288 FE04            LTDN:         CP 4
03640 D28A 2004                     JR   NZ,RTDN
03650 D28C 3E9A                     LD   A,$9A                ; A <- "┌"
03660 D28E 180A                     JR   DNOB
03670 D290 FE06            RTDN:         CP 6
03680 D292 2004                     JR   NZ,DNDN
03690 D294 3E97                     LD   A,$97                ; A <- "┐"
03700 D296 1802                     JR   DNOB
03710 D298 3E91            DNDN:         LD A,$91             ; A <- "!"
03720 D29A 320EE0          DNOB:         LD  (ORBIT),A
03730 D29D                                                    ;
03740 D29D 3A0DE0                   LD   A, (TS)
03750 D2A0 FE00                     CP   0
03760 D2A2 200B                     JR   NZ,SKDNBK
03770 D2A4 3E6E            TRDNBK:       LD A,110             ; TRON down bike
03780 D2A6 320FE0                   LD   (BIKE),A
03790 D2A9 3E02                     LD   A,2
03800 D2AB 3204E0                   LD   (TRMV),A             ; (TRMV) <- 2
03810 D2AE C9                       RET
03820 D2AF 3ED2            SKDNBK:       LD A,210             ; SARK down bike
03830 D2B1 320FE0                   LD   (BIKE),A
03840 D2B4 3E02                     LD   A,2
03850 D2B6 3205E0                   LD   (SKMV),A             ; (SKMV) <- 2
03860 D2B9 C9                       RET
03870 D2BA                                                    ;
03900 D2BA 3A0DE0          CHRLT:        LD A, (TS)           ; character left
03910 D2BD FE00                     CP   0
03920 D2BF 2005                     JR   NZ,SKLTOB
```

```
03930 D2C1 3A04E0      TRLTOB:         LD A,(TRMV)
03940 D2C4 1803                 JR     UPLT
03950 D2C6 3A05E0      SKLTOB:         LD A,(SKMV)
03960 D2C9 FE08        UPLT:           CP 8
03970 D2CB 2004                 JR     NZ,DNLT
03980 D2CD 3E97                 LD     A,$97            ; A <- "┐"
03990 D2CF 180A                 JR     LTOB
04000 D2D1 FE02        DNLT:           CP 2
04010 D2D3 2004                 JR     NZ,LTLT
04020 D2D5 3E98                 LD     A,$98            ; A <- "┘"
04030 D2D7 1802                 JR     LTOB
04040 D2D9 3E90        LTLT:           LD A,$90         ; A <- "─"
04050 D2DB 320EE0      LTOB:           LD (ORBIT),A
04060 D2DE                                              ;
04070 D2DE 3A0DE0               LD     A,(TS)
04080 D2E1 FE00                 CP     0
04090 D2E3 200B                 JR     NZ,SKLTBK
04100 D2E5 3E78        TRLTBK:         LD A,120         ; TRON left bike
04110 D2E7 320FE0               LD     (BIKE),A
04120 D2EA 3E04                 LD     A,4
04130 D2EC 3204E0               LD     (TRMV),A         ; (TRMV) <- 4
04140 D2EF C9                   RET
04150 D2F0 3EDC        SKLTBK:         LD A,220         ; SARK left bike
04160 D2F2 320FE0               LD     (BIKE),A
04170 D2F5 3E04                 LD     A,4
04180 D2F7 3205E0               LD     (SKMV),A         ; (SKMV) <- 4
04190 D2FA C9                   RET
04200 D2FB                                              ;
04210 D2FB 3A0DE0      CHRRT:          LD A,(TS)        ; character right
04220 D2FE FE00                 CP     0
04230 D300 2005                 JR     NZ,SKRTOB
04240 D302 3A04E0      TRRTOB:         LD A,(TRMV)
04250 D305 1803                 JR     UPRT
04260 D307 3A05E0      SKRTOB:         LD A,(SKMV)
04270 D30A FE08        UPRT:           CP 8
04280 D30C 2004                 JR     NZ,DNRT
04290 D30E 3E9A                 LD     A,$9A            ; A <- "┌"
04300 D310 180A                 JR     RTOB
04310 D312 FE02        DNRT:           CP 2
04320 D314 2004                 JR     NZ,RTRT
04330 D316 3E99                 LD     A,$99            ; A <- "└"
04340 D318 1802                 JR     RTOB
04350 D31A 3E90        RTRT:           LD A,$90         ; A <- "─"
04360 D31C 320EE0      RTOB:           LD (ORBIT),A
04370 D31F                                              ;
04380 D31F 3A0DE0               LD     A,(TS)
04390 D322 FE00                 CP     0
04400 D324 200B                 JR     NZ,SKRTBK
04410 D326 3E82        TRRTBK:         LD A,130         ; TRON right bike
04420 D328 320FE0               LD     (BIKE),A
```

```
04430 D32B 3E06                        LD    A,6
04440 D32D 3204E0                      LD    (TRMV),A                  ; (TRMV) <- 6
04450 D330 C9                          RET
04460 D331 3EE6          SKRTBK:             LD A,230                  ; SARK right bike
04470 D333 320FE0                      LD    (BIKE),A
04480 D336 3E06                        LD    A,6
04490 D338 3205E0                      LD    (SKMV),A                  ; (SKMV) <- 6
04500 D33B C9                          RET
04510 D33C                                                             ;
04515 D33C                                                             ;
04520 D33C 7A            PRINT:              LD A,D                    ; print sub ■■■■
04530 D33D FE01                        CP    1
04540 D33F 2004                        JR    NZ,OBATR
04550 D341 3E27          BKATR:              LD A,$27                  ; COLOR 7, CGEN 1 (bike)
04560 D343 181F                        JR    ATROUT
04570 D345 3A0DE0        OBATR:              LD A,(TS)
04580 D348 FE00                        CP    0
04590 D34A 200D                        JR    NZ,SKATR
04600 D34C 7A            TRATR:              LD A,D
04610 D34D FE00                        CP    0
04620 D34F 2004                        JR    NZ,TRCRSH
04630 D351 3E26                        LD    A,$26                     ; COLOR 6, CGEN 1 (TR orbit)
04640 D353 180F                        JR    ATROUT
04650 D355 3E06          TRCRSH:             LD A,$06                  ; COLOR 6, CGEN 0 (TR X)
04660 D357 180B                        JR    ATROUT
04670 D359 7A            SKATR:              LD A,D
04680 D35A FE00                        CP    0
04690 D35C 2004                        JR    NZ,SKCRSH
04700 D35E 3E22                        LD    A,$22                     ; COLOR 2, CGEN 1 (SK orbit)
04710 D360 1802                        JR    ATROUT
04720 D362 3E02          SKCRSH:             LD A,$02                  ; COLOR 2, CGEN 0 (SK X)
04730 D364                                                             ;
04740 D364 CBA0          ATROUT:             RES 4,B                   ; attribute address
04750 D366 ED79                        OUT   (C),A                     ; attribute out
04760 D368 CBE0          CHROUT:             SET 4,B                   ; text VRAM address
04770 D36A 7B                          LD    A,E                       ; A <- character code
04780 D36B ED79                        OUT   (C),A                     ; character out
04790 D36D C9                          RET
04800 D36E                                                             ;
04810 D36E                                                             ; ■■■ Character reverse ■■■
04820 D36E                                                             ;
04830 D36E 010020        CREV:               LD BC,$2000               ; attribute top address
04840 D371 ED78          LPREV:              IN A,(C)
04850 D373 CBDF                        SET   3,A                       ; CREV flag set
04860 D375 ED79                        OUT   (C),A                     ; attribute out
04870 D377 03                          INC   BC
04880 D378 78                          LD    A,B
04890 D379 FE23                        CP    $23
04900 D37B 38F4                        JR    C,LPREV
04910 D37D 79                          LD    A,C
```

```
04920 D37E FEE8                     CP     $E8                  ; < 23E8H ?
04930 D380 38EF                     JR     C,LPREV
04940 D382 C9                       RET
04950 D383                                                      ;
60000 D383                          END
```

付録6　X1，X1C，X1D
マニュアル対応表

本文ページ	X1 マニュアルページ	X1C マニュアルページ	X1D マニュアルページ
2	175	200	224
2	195	220	※
4	107	112	120
5	175～177	200～202	224～226
38	176	201	225
42	61	62	64
74	91	94	102
76	72	72	80
81	168	192～	216～
82	62	63	65
117	176	201	225
125	176	201	225
125	183	209	236
127	41	42	44
127	20	20	22
144	178	203～204	227～229
146	178	203～204	227～229
152	183	209	236
201	34	34	36
211	154	162	170

※　X1DのマニュアルにはBASICシステムテープのコピープログラムは掲載されておりません。

≪さくいん≫

著者紹介

清水　保弘（しみず　やすひろ）

1954年、横浜市生れ。東京大学理学部数学科卒業。現在、東京都立大学大学院博士課程に在学中。専攻は数学（幾何学、群論)。数学研究におけるパソコン利用の可能性について追究している

誰にでもわかる

X1 マシン語入門

著者　清水　保弘　　　　定価 2,800円

昭和59年 3 月 1 日　　第 2 刷発行

印刷・製本　　昭和工芸印刷㈱
（234）1801

㈱日本ソフト&ハード社
出版部　☎03(209)2585
コンピュータ・イレブン